HAL
open science

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. [Research Report] LaBRI, CNRS UMR 5800. 2019. hal-02376832v1

HAL Id: hal-02376832

https://hal.science/hal-02376832v1

Submitted on 22 Nov 2019 (v1), last revised 15 May 2020 (v3)

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
Sorbonne Université, Paris, LIP6, UMR 7606, France
anais.durand@lip6.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 message-passing systems. We first reformulate the definition of self-stabilization to accommodate with the highly dynamic context. Then, we address the self-stabilizing leader election problem for three wide classes of time-vaying graphs (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, where $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$.

In more detail, we 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. The first one stabilizes in at most 3Δ rounds; the stabilization time of the two others cannot be bounded in general, however we exhibit small time complexity bounds holding for them in more favorable cases, meaning that these solutions are speculative. Precisely, in Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$, both algorithms stabilize in $O(\Delta)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

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

Funding This study has been partially supported by the ANR project Estate (ANR-16-CE25-0009).

1 Introduction

Context and related work. Starting from an arbitrary configuration, a *self-stabilizing algorithm* [13] 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 these latter last a finite time (as opposed to crash failures, for example) and their frequency is assumed to be 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 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 be still achieved despite such faults occur. Indeed, strong forms of self-stabilization have been proposed to tolerate permanent failures, *e.g.*, *fault-tolerant self-*

stabilization [3] to cope with process crashes, and *strict stabilization* [21] to withstand Byzantine failures. Furthermore, several self-stabilizing algorithms, *e.g.*, [11, 12], withstand intermittent failures such as frequent lost, duplication, or reordering of messages, meaning their convergence is still effective despite such faults continue to 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. In particular, several approaches, like *superstabilization* [16] and *gradual stabilization* [1], have been proposed to efficiently tolerate topological changes when they are both spatially and timely sparse. However, these approaches 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. Ensuring convergence in such networks regardless the initial configuration may seem to be very challenging, even impossible in many cases [5]. However, notice that a recent work [4] deals with the self-stabilizing exploration of a highly dynamic ring by a cohort of synchronous robots equipped of visibility sensors, moving actuators, yet no communication capabilities. Yet, self-stabilization still needs to be investigated in the context of *highly dynamic message-passing networks*.

Several works aim at proposing a general graph-based model to capture the network dynamics. In [22], the network dynamics is represented as a sequence of graphs called *evolving graphs*. In [8], 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). Still in [8], TVGs are gathered and ordered into classes according to the temporal characteristics of edge presence. Such taxonomy allows to provide lower and upper bounds *w.r.t.* the distributed tasks to be solved.

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* [20] is another possible approach for 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 improbable) 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.

Contribution. We initiate research on self-stabilization in the context of *highly dynamic message-passing systems*. We model the network dynamics using the TVG paradigm.

We first reformulate the definition of self-stabilization to accommodate with the highly dynamic context. We then propose self-stabilizing leader election algorithms for three wide classes of TVGs respectively denoted by $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$. $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is the class of TVGs with temporal diameter bounded by Δ (introduced in [18]), $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ is the class of TVGs with temporal diameter quasi-bounded by Δ (introduced in the present paper), and $\mathcal{TC}^{\mathcal{R}}$ is the class of TVGs with recurrent temporal connectivity (introduced in [8]). Notice that, contrary to [4], the three classes studied here never enforce the network to be in a particular topology at a given time.

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. Then, we propose a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ assuming every process knows Δ and n (the number of processes). The stabilization time of this algorithm cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, however the algorithm is speculative in the sense that 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 Δ is unknown by processes, while n still is. Notice in particular that finding a self-stabilizing solution in this class was rather challenging, since in this class there is no timeliness guarantee at all. Again, the stabilization time of the algorithm cannot be bounded in $\mathcal{TC}^{\mathcal{R}}$, however the algorithm is speculative in the sense that its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most $\Delta + 1$ rounds.

An overview of the properties of the proposed algorithms is presented in the table 1.

TVG	Knowledge		Alg.	Stabilization time	Msg size	Remarks
	Δ	n				
$\mathcal{TC}^{\mathcal{B}}(\Delta)$	✓	×	1	3Δ	$O(\log(n+\Delta))$	Δ bounded timestamps
	✓	✓	2	2Δ	$O(n(\log(n+\Delta)))$	Δ bounded timestamps
	×	✓	3	$\Delta+1$	unbounded	unbounded timestamps
$\mathcal{TC}^{\mathcal{Q}}(\Delta)$	✓	✓	2	unbounded	$O(n(\log(n+\Delta)))$	Δ bounded timestamps
$\mathcal{TC}^{\mathcal{R}}$	×	✓	3	unbounded	unbounded	unbounded timestamps

Table 1 : Overview of self-stabilizing leader election algorithms

Roadmap. The remainder of the paper is organized as follows. In the next section, we first define TVGs and their related concepts, and then the computational model. The three next sections are dedicated to the three TVG classes in which we investigate self-stabilizing solutions to the leader election problem. We give some perspectives in the last section.

2 Preliminaries

2.1 Time-varying Graphs

A *time-varying graph* (TVG for short) [8] 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 $\mathbb{N}^*$ called the *lifetime* of $\mathcal{G}$, and $\rho : E \times \mathcal{T} \to \{0, 1\}$ is the *presence* function that indicates whether a given arc exists at a given time. Notice that our definition of TVG is close to the model, called *evolving graphs*, defined in [22].

Let $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ be a TVG. 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 $[t, t']$, noted $\mathcal{G}_{[t,t']}$, 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.

2.2 TVG Classes

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

Class $\mathcal{TC}$ (Temporal Connectivity): Every node can reach all the others at least once through a journey. Formally,

$$\mathcal{G} \in \mathcal{TC} \text{ if } \forall p, q \in V, p \rightsquigarrow q.$$

This class is denoted by $\mathcal{C}_3$ in [8] and $\mathcal{F}_2$ in [7].

Class $\mathcal{TC}^{\mathcal{R}}$ (Recurrent Temporal Connectivity): At any point in time, every node can reach all the others through a journey. Formally,

$$\mathcal{G} \in \mathcal{TC}^{\mathcal{R}} \text{ if } ITVG(\mathcal{G}) \wedge \forall t \in \mathcal{T}, \mathcal{G}_{[t,+\infty)} \in \mathcal{TC}.$$

This class is denoted by $\mathcal{C}_5$ in [8].

Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Bounded Temporal Diameter): 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,

$$\mathcal{G} \in \mathcal{TC}^{\mathcal{B}}(\Delta) \text{ if } ITVG(\mathcal{G}) \wedge \forall t \in \mathcal{T}, \mathcal{G}_{[t,t+\Delta)} \in \mathcal{TC}.$$

This class is denoted by $\mathcal{TC}(\Delta)$ in [18].

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,

$$\mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta) \text{ if } ITVG(\mathcal{G}) \wedge \forall p, q \in V, \forall t \in \mathcal{T}, \exists t' \geq t - 1, \hat{d}_{p,t'}(q) \leq \Delta.$$

$$\begin{tikzcd} & \mathcal{TC}^{\mathcal{Q}}(\Delta) \arrow[dr] & & \\ \mathcal{TC}^{\mathcal{B}}(\Delta) \arrow[ur] \arrow[rr] & & \mathcal{TC}^{\mathcal{R}} \arrow[r] & \mathcal{TC} \end{tikzcd}$$

Figure 1 Hierarchy of the TVG classes considered in this paper (the sub-hierarchy inside the rectangle comes from [6]).

▶ **Theorem 1.** $\forall \Delta \in \mathbb{N}^*$, $\mathcal{TC}^{\mathcal{Q}}(\Delta) \nsubseteq \mathcal{TC}^{\mathcal{B}}(\Delta)$.

Proof. Let $\mathcal{G}$ be the TVG of two nodes p and q such that p and q are connected at time 2^i, for every $i \geq 0$. There is an infinite number of points in time where there is a journey of length $1 \leq \Delta$ between p and q so $\mathcal{G} \in \mathcal{TC}^{\mathcal{Q}}(\Delta)$ but the temporal diameter cannot be bounded so $\mathcal{G} \notin \mathcal{TC}^{\mathcal{B}}(\Delta)$ whatever the value of Δ is. Hence, $\mathcal{TC}^{\mathcal{Q}}(\Delta) \nsubseteq \mathcal{TC}^{\mathcal{B}}(\Delta)$. ◀

Figure 1 illustrates the hierarchy between those classes. An arrow from class A to class B implies that $A \subseteq B$. $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$

2.3 Computational Model

We consider the computational model defined in [2, 9]. We assume a *distributed system* made of n *processes*, each of them having 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 noted $id(p)$ and taken in an arbitrary domain $IDSET$ totally ordered by $<$. As commonly done in the literature, we assume that each identifier can be stored using $\Theta(\log n)$ bits. In the sequel, we denote by ℓ the process of minimum identifier. Processes are assumed to communicate through an interconnected network that evolves over the time. Hence, the topology of the network is conveniently modeled by an infinite TVG $\mathcal{G} = (V, E, \mathcal{T}, \rho)$, where nodes represent processes.

Configurations, rounds, and executions. The *state* of a process is defined by the values of its variables. A *configuration* (of the system) is a vector of n components $(s_1, s_2, \ldots, s_n)$ where s_1 to s_n represent the states of the n processes in V. Processes communicate by exchanging messages and proceed in *synchronous rounds*. For every $i \geq 1$, the communication network at Round i is defined by $G_{o_{\mathcal{T}}+i-1}$, *i.e.*, the snapshot of $\mathcal{G}$ after $i-1$ rounds since 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. Notice that $\mathcal{N}(p)^i$ is assumed to be unknown by process p, whatever the value of i is.

Let γ_0 be the initial configuration of the system. For any 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 performed as follows:

1. Every process 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.

These three steps are described in the *local algorithm* of every process p. A distributed algorithm is a collection of n local algorithms, one per process.

In this context, an *execution* of a distributed algorithm $\mathcal{A}$ in $\mathcal{G}$ is an infinite sequence of configurations $\gamma_0, \gamma_1, \ldots$ 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 $G_{o_{\mathcal{T}}+i-1}$.

Stabilizing systems. Self-stabilization has been originally defined for static networks. In [13, 15], 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. Following [15], we accommodate this concept with highly dynamic environments by splitting the definition into two properties. The *convergence* property requires every execution of the algorithm in the considered system to eventually reach a legitimate configuration. The *correctness* property requires every possible execution suffix starting from a legitimate configuration to satisfy the specification.

In [13, 15], self-stabilization is defined as follows: an algorithm is self-stabilizing if, starting from an arbitrary configuration, it makes the system converge to a configuration from which every possible execution suffix satisfies the intended specification.

▶ **Definition 2** (Self-stabilization). *An algorithm $\mathcal{A}$ is* self-stabilizing *for the specification SP on the class $\mathcal{C}$ of infinite TVGs if there exists a non-empty subset of configurations $\mathcal{L}$, called the set of* legitimate configurations*, such that:*

1. *for every $\mathcal{G} \in \mathcal{C}$, for every configuration γ, every execution of $\mathcal{A}$ in $\mathcal{G}$ starting from γ contains a legitimate configuration $\gamma' \in \mathcal{L}$ (*Convergence*), and*
2. *for every $\mathcal{G} \in \mathcal{C}$, for every $t \geq o_{\mathcal{T}}$, for every legitimate configuration $\gamma \in \mathcal{L}$, for 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.*

Often self-stabilization requires the closure of $\mathcal{L}$. Meaning that any step from a configuration of $\mathcal{L}$ has to reach a configuration that is also legitimate. Nevertheless, some authors define self-stabilization without including this property [15, 14]. Even without the closure of $\mathcal{L}$, self-stabilization ensures the closure of SP : any execution starting from a configuration of $\mathcal{L}$ satisfies SP. So any self-stabilizing algorithm is pseudo-stabilizing but the converse is not true. As the knowledge of the current configuration of a pseudo-stabilizing algorithm cannot predict if the end of the execution satisfies SP.

Let $\mathcal{C}$ be a class of infinite TVGs, we say that $\mathcal{C}$ is *recurring* if $\forall \mathcal{G} \in \mathcal{C}$, for every $t \geq o_{\mathcal{T}}$, $\mathcal{G}_{[t,+\infty)} \in \mathcal{C}$. The three classes we will consider (*i.e.*, $\mathcal{TC}^{\mathcal{R}}$, $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$) are recurring. In this case, the definition of the self-stabilization can be slightly simplified as follows.

▶ Remark 3. An algorithm $\mathcal{A}$ is *self-stabilizing* for the specification SP on the recurring class $\mathcal{C}$ of infinite TVGs if there exists a non-empty subset of legitimate configurations $\mathcal{L}$ such that:

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

2.4 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, noted here $lid(p)$.

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 $\mathcal{C}$.

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

We propose a self-stabilizing algorithm (see Algorithm 1) for TVGs with bounded temporal diameter, *i.e.*, Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$. The bound on the temporal diameter, Δ, is known by all processes.

Algorithm 1. In this algorithm, each process p maintains two variables: the output $lid(p)$ will eventually contain the ID of the leader and $ttl(p)$ 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 Lines u1-u3).

At each round i, p first sends its leader ID together with its degree of mistrust (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, see Line 7), if any. If id is smaller than $lid(p)$, p updates its leader $lid(p)$ (see Lines 8-9). 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 Lines 11-12).

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 Lines 15-17). 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 never more reset, ensuring that ℓ will remain the leader forever.

Self-stabilization and complexity.

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

▶ Remark 4. 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 5.** *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.

Base case: The case $i = 1$ is trivial, since by definition $ttl(p) \geq 0$.

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

```
Inputs:
  Δ ∈ ℕ*            :  upper bound on the temporal diameter
  id(p) ∈ IDSET     :  ID of p

Local variables:
  lid(p) ∈ IDSET                  :  ID of the leader
  ttl(p) ∈ {0, ..., 2Δ − 1}       :  degree of mistrust in lid(p)

Macros:
updateTTL(v):
u1:  if v ≥ 2Δ then // Reset
u2:      lid(p) := id(p)
u3:      ttl(p) := 0
u4:  else if lid(p) ≠ id(p) then  ttl(p) := v

1:  Repeat Forever
2:      SEND(⟨lid(p), ttl(p)⟩)
3:      mailbox := RECEIVE()
4:      if mailbox = ∅ then
5:          updateTTL(ttl(p) + 1)
6:      else
7:          ⟨lid, ttl⟩ := min{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) ≥ id(p) then // Reset
16:         lid(p) := id(p)
17:         ttl(p) := 0
```

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, 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 10 or 12). 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. Similarly to Case 1, $ttl(p) \geq i - 1$ at the beginning of Round i. ◀

Note that Lemma 5 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).

▶ **Corollary 6.** *At the end of Round 2Δ, the configuration is quasi-legitimate.*

Proof. By Lemma 5 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 4, 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 the set of quasi-legitimate configuration is *closed*. Recall that a set of configuration S is closed if every step of the algorithm starting in a configuration of S leads to a configuration of S.

▶ **Lemma 7.** *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 4, we conclude that $lid(\ell) = id(\ell)$ and $ttl(\ell) = 0$ in γ'. Hence, γ' is quasi-legitimate, *i.e.*, the set of quasi-legitimate configuration is closed and we are done. ◀

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 8.** *Let $\mathcal{G}$ be a TVG of Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Let $t \geq o_{\mathcal{T}}$. Let 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 following 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 the 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, the base case is immediate since the initial configuration is quasi-legitimate (by hypothesis) and every subsequent configuration is quasi-legitimate too (by Lemma 7).

Induction step: Consider any process p such that $\hat{d}_{\ell,t+i-1}(p) \leq d$ with $d > 0$. If $\hat{d}_{\ell,t+i-1}(p) < d$, then 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 $\langle id(\ell), tll_q \rangle$ to p during Round $i + d$ with $tll_q \leq d - 1$, where tll_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 7) the minimum message received by p in Round $i + d$ is $\langle id(\ell), ttl \rangle$ with $ttl \leq tll_q \leq d - 1$. 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 (see Lines 4-14) and p does not reset (Lines u1-u3). 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 *(*)* 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 4. Hence, the configuration at the end of Round r is legitimate. ◀

As direct consequence of Corollary 6 and Lemma 8, we obtain the convergence.

▶ **Corollary 9.** *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 10.** *Let $\mathcal{G}$ be a TVG of $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Let $t \geq o_{\mathcal{T}}$. Let 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 is such that for every process p, $lid(p) = id(\ell)$ and $ttl(p) \leq \Delta + r$.*

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 $i \in \{2, ..., \Delta - 1\}$. 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. ◀

By Remark 3 (page 6), the correctness part of the self-stabilization of our algorithm can be established by the following theorem.

▶ **Theorem 11.** *For every $\mathcal{G} \in \mathcal{TC}^{\mathcal{B}}(\Delta)$, for every legitimate configuration γ, for the execution of Algorithm 1 in $\mathcal{G}$ starting from γ, SP_{LE} holds.*

Proof. Let $e = \gamma_0 ... \gamma_i ...$ be an execution of Algorithm 1 in $\mathcal{G}$ starting from a legitimate configuration. First, as γ_0 is legitimate, we have $lid(p) = id(\ell)$, for every process p (by definition). Then, by Lemma 10, 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 8 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 and we are done. ◀

By Corollary 9 and Theorem 11, follows.

▶ **Corollary 12.** *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 $O(\log(n + \Delta))$ bits per process and messages of size $O(\log(n + \Delta))$ bits.*

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

We propose a self-stabilizing algorithm (see Algorithm 2) for TVGs with quasi bounded temporal diameter, *i.e.*, Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. Δ and the exact number of processes n are known by all processes.

Algorithm 2. In this algorithm, 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 4-6).

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 (Lines i1-i2), and in either case, $\langle id, t\rangle$ is appended at the tail of the queue (Lines i3 and i7). In particular, since the size of $members(p)$ is limited, if the queue is full, then the head of the queue is removed to make room for the new value (Lines i5-i6). 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 7-8) and then, $\langle id(p), 0\rangle$ is systematically inserted at the tail of the queue (Line 9). 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 are 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 10). Hence, the process of lowest ID, ℓ, is eventually elected.

Self-stabilization.

▶ **Lemma 13.** *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 $i \geq 1$.

Base case: The case $i = 1$ is trivial since $members(p)[f]$ is a natural integer.

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 i3 or i7. In both cases, after executing Line 8, $members(p)[f] \geq i-1$, 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 8 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

Algorithm 2: Self-stabilizing leader election for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ with Δ known, 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** $\exists t', \left\langle id, t'\right\rangle \in members(p)$ **then**
i2: remove $\left\langle id, t'\right\rangle$ from $members(p)$
i3: push $\left\langle id, \min(t, t')\right\rangle$ at the tail of $members(p)$
i4: **else**
i5: **if** $|members(p)| = n$ **then**
i6: remove the head of $members(p)$
i7: 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**
6: $insert(p, \langle id, t\rangle)$
7: **forall** $\langle id, t\rangle \in members(p) : t < \Delta$ **do**
8: $members(p)[id] + +$
9: $insert(p, \langle id(p), 0\rangle)$
10: $lid(p) := \min\{id \ : \ \langle id, _\rangle \in members(p)\}$

Line i3, $members(p)[f] := min(t, members(p)[f])$. 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 the following corollary.

▶ **Corollary 14.** *In any round $\Delta + i$ with $i \geq 1$, no process receives a message containing fake IDs.*

▶ **Lemma 15.** *$\forall p, q \in V$, 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 i3),
- or Line i6 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 real IDs of processes in V since p will not receive any fake ID (Corollary 14). Thus, at most $n-1$ IDs different than 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 nodes 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 16.** *$\forall p, q \in V$, 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 9.

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 $i \in \{1,...,k\}$

- $id(q)$ is forever in $members(p_i)$ by the end Round τ_i
- $members(p_i)[q] \leq \tau_i - \tau_1 + 1$ 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 insert $id(q)$ 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 15. Still following the algorithm, $members(p_1)[q] = 1$ at the end of Round τ_1, p_1, and we are done.

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 15 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 $\tau_k \leq t(q,p) + \Delta$. ◀

We define a *legitimate* configuration of Algorithm 2 as any configuration 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 is closed. 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 $lid(p)$ is computed as the minimum of this set, *i.e.* $id(\ell)$, forever. Hence the following lemma.

▶ **Lemma 17.** *Every execution of Algorithm 2 that starts from a legitimate configuration in an arbitrary TVG satisfies SP_{LE}.*

▶ **Lemma 18.** *$\exists t \geq \Delta$ such that the configuration reached at the end of Round $t + \Delta$ is legitimate.*

Proof. Corollary 14 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 16, 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 i5), $members(p)$ is exactly the set of IDs of every process. By Line 10, we also have $lid(p) = id(\ell)$. ◀

▶ **Theorem 19.** *Algorithm 2 is a self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. It requires $O(n(\log(n+\Delta)))$ bits per process and messages of size $O(n(\log(n+\Delta)))$ bits.*

Speculation. The stabilization time of Algorithm 2 cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. 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 convergence of Algorithm 2 not bounded.

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.

Since $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta)$ the previous proof holds for $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Yet, for every processes p and q, $t(q,p)$ is exactly Δ in this class of TVGs. Hence in the proof of Lemma 18, we have $T = \max\{t(q,p) : q,p \in V\} = \Delta$; this ensures that in Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$, the configuration reached at the end of Round 2Δ is legitimate.

▶ **Theorem 20.** *The stabilization time of Algorithm 2 in Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most 2Δ rounds.*

5 Class $\mathcal{TC}^{\mathcal{R}}$

Algorithm 3. Similarly to Algorithm 2, in this algorithm, 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 $members(p)$ (Lines i6-i7). This overwriting mechanism allows to eventually remove all fake IDs from $members(p)$, since their timestamps regularly increase.

After $members(p)$ has been updated, all timestamps of $members(p)$ are incremented (Lines 6-7) and then, $\langle id(0), 0\rangle$ is systematically inserted in the map (Line 8).

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 21. Second, by definition of $\mathcal{TC}^{\mathcal{R}}$, at any point in time, every node 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 22. 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 9). 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 ∈ ℕ            :  number of processes
  id(p) ∈ IDSET    :  ID of p

Local variables:
  members(p)       :  map of size at most n
                      contains pairs ⟨id, t⟩ ∈ IDSET × ℕ
  lid(p) ∈ IDSET   :  ID of the leader

Macros:
max(p):
m1:  if |members(p)| < n then return ⊥
m2:  else return ⟨id, t⟩ ∈ members(p) with maximum timestamp t

insert(p, ⟨id, t⟩):
i1:  if ⟨id, _⟩ ∈ members(p) then members(p)[id] := min(t, members(p)[id])
i2:  else if max(p) = ⊥ then add ⟨id, t⟩ in members(p)
i3:  else
i4:      ⟨idM, tM⟩ := max(p)
i5:      if t < tM then
i6:          remove ⟨idM, tM⟩ from members(p)
i7:          add ⟨id, t⟩ in members(p)

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

Self-stabilization.

▶ **Lemma 21.** *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 $members(p)[f] \geq i - 1$.*

This lemma and its proof are identical to Lemma 13 of Algorithm 2, page 11.

▶ **Lemma 22.** *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 8). Since the timestamp associated with every other ID in $members(p)$ has been incremented beforehand (line 7), $\langle p, 0\rangle \in members(p)$ 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 21, the timestamps associated to fake IDs are greater than or equal to $i-1$. Now, timestamps are incremented during Round i (Line 7), 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 7), hence $members(q)[p] \leq i-1$ at the end of Round i. ◀

Similarly to Algorithm 2, we define a *legitimate* configuration of Algorithm 3 as any configuration where for every process p, we have $lid(p) = id(\ell)$ and $\{id : \langle id, _\rangle \in members(p)\} = \{id(q) : q \in V\}$. First, 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. Hence, the set of legitimate configurations is closed and so we have:

▶ **Lemma 23.** *Every execution of Algorithm 3 that starts from a legitimate configuration in an arbitrary TVG satisfies SP_{LE}.*

▶ **Theorem 24.** *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 22. 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 23, we are done. ◀

Speculation. Similarly to $\mathcal{TC}^\mathcal{Q}(\Delta)$, stabilization time cannot be bounded in $\mathcal{TC}^\mathcal{R}$ (*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 being unknown by processes.

▶ **Theorem 25.** *The stabilization time of Algorithm 3 in $\mathcal{TC}^\mathcal{B}(\Delta)$ is at most $\Delta+1$ rounds.*

The proof is the same as the one of Theorem 24 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 $\max_{p\in V} \delta(p)+1 = \Delta+1$.

6 Conclusion

We initiated research on self-stabilization in highly dynamic message-passing systems by proposing self-stabilizing leader election algorithms for three major classes of time-varying graphs. Beyond extending our results to ever more general classes, our future work will focus on studying expressiveness of self-stabilization in TVGs. As a matter of fact, we plan to investigate broadcast problems, again in very general TVG classes. Indeed, coupled with our leader election solutions, they should allow to build generic transformers [10, 19].

References

1. Karine Altisen, Stéphane Devismes, Anaïs Durand, and Franck Petit. Gradual stabilization. *JPDC*, 123:26–45, 2019.
2. 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.
3. Joffroy Beauquier and Synnöve Kekkonen-Moneta. On FTSS-solvable distributed problems. In *PODC97, the 16th Annual ACM Symposium on Principles of Distributed Computing*, page 290, 1997.
4. Marjorie Bournat, Ajoy K. Datta, and Swan Dubois. Self-stabilizing robots in highly dynamic environments. *Theor. Comput. Sci.*, 772:88–110, 2019.
5. 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.
6. Arnaud Casteigts. *A Journey through Dynamic Networks (with Excursions)*. HDR, Université de Bordeaux, 2018.
7. Arnaud Casteigts, Serge Chaumette, and Afonso Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *SIROCCO*, volume 5869, pages 126–140, 2009.
8. 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.
9. Bernadette Charron-Bost and Shlomo Moran. The firing squad problem revisited. In *STACS*, pages 20:1–20:14, 2018.
10. 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.
11. Ajoy K. Datta, Stéphane Devismes, Lawrence L. Larmore, and Vincent Villain. Self-stabilizing weak leader election in anonymous trees using constant memory per edge. *PPL*, 27(2):1–18, 2017.
12. Sylvie Delaët, Bertrand Ducourthial, and Sébastien Tixeuil. Self-stabilization with r-operators revisited. *Journal of Aerospace Computing, Information, and Communication (JACIC)*, 3(10):498–514, 2006.
13. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
14. S. Dolev, S. Dubois, M. Potop-Butucaru, and S. Tixeuil. Stabilizing data-link over non-FIFO channels with optimal fault-resilience. *Information Processing Letters*, 111(18):912–920, 2011.
15. Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.
16. Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 1995.
17. S. Dubois and R. Guerraoui. Introducing speculation in self-stabilization: an application to mutual exclusion. pages 290–298, 2013.
18. 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.

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

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

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

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