Information and Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Self-stabilizing leader election in polynomial steps ☆,☆☆

Karine Altisen [a], Alain Cournier [b], Stéphane Devismes [a], Anaïs Durand [a,*], Franck Petit [c]

[a] *VERIMAG UMR 5104, Université Grenoble Alpes, France*
[b] *MIS Lab., Université Picardie Jules Verne, France*
[c] *LIP6 UMR 7606, INRIA, UPMC Sorbonne Universités, France*

ARTICLE INFO

Article history:
Received 15 January 2015
Available online xxxx

Keywords:
Distributed algorithms
Fault-tolerance
Self-stabilization
Leader election
Unfair daemon

ABSTRACT

We propose a silent self-stabilizing leader election algorithm for bidirectional arbitrary connected identified networks. This algorithm is written in the locally shared memory model under the distributed unfair daemon. It requires no global knowledge on the network. Its stabilization time is in $\Theta(n^3)$ steps in the worst case, where n is the number of processes. Its memory requirement is asymptotically optimal, *i.e.*, $\Theta(\log n)$ bits per processes. Its round complexity is of the same order of magnitude — *i.e.*, $\Theta(n)$ rounds — as the best existing algorithms designed with similar settings. To the best of our knowledge, this is the first self-stabilizing leader election algorithm for arbitrary identified networks that is proven to achieve a stabilization time polynomial in steps. By contrast, we show that the previous best existing algorithms designed with similar settings stabilize in a non-polynomial number of steps in the worst case.

1. Introduction

In distributed computing, the *leader election* problem consists in distinguishing a single process, so-called the leader, among the others. We consider here identified networks. So, as it is usually done, we augment the problem by requiring all processes to eventually know the identifier of the leader. The leader election is fundamental as it is a basic component to solve many other important problems, *e.g.*, consensus, spanning tree constructions, implementing broadcasting and convergecasting methods, *etc.*

Self-stabilization [4,5] is a versatile technique to withstand *any* transient fault in a distributed system: a self-stabilizing algorithm is able to recover, *i.e.*, reach a legitimate configuration, in finite time, regardless the *arbitrary* initial configuration of the system, and therefore also after the occurrence of transient faults. Thus, self-stabilization makes no hypotheses on the nature or extent of transient faults that could hit the system, and recovers from the effects of those faults in a unified manner. Such versatility comes at a price. After transient faults, there is a finite period of time, called the *stabilization phase*, before the system returns to a legitimate configuration. The *stabilization time* is then the worst case duration of the

☆ A preliminary version of this paper has been published in SSS'2014 [3].

☆☆ This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program Investissement d'avenir and the AGIR project DIAMS.

* Corresponding author.
E-mail addresses: karine.altisen@imag.fr (K. Altisen), alain.cournier@u-picardie.fr (A. Cournier), stephane.devismes@imag.fr (S. Devismes), anais.durand@imag.fr (A. Durand), franck.petit@lip6.fr (F. Petit).

http://dx.doi.org/10.1016/j.ic.2016.09.002
0890-5401/

stabilization phase, *i.e.*, the maximum time to reach a legitimate configuration starting from an arbitrary one. Notice that efficiency of self-stabilizing algorithms is mainly evaluated according to their stabilization time and memory requirement.

We consider deterministic[1] asynchronous silent self-stabilizing leader election problem in bidirectional, connected, and identified networks of arbitrary topology. We investigate solutions to this problem which are written in the locally shared memory model introduced by Dijkstra [4]. In this model, the distributed unfair daemon is known as the weakest scheduling assumption. Under such an assumption, proving that a given algorithm is self-stabilizing implies that the stabilization time must be finite in terms of atomic steps. However, despite some solutions assuming all these settings (in particular the unfairness assumption) are available in the literature [1,2,6], none of them is proven to achieve a polynomial upper bound in steps on its stabilization time. Actually, the time complexities of all these solutions are analyzed in terms of rounds only.

Related work. In [7], Dolev et al. showed that silent self-stabilizing leader election requires $\Omega(\log n)$ bits per process, where n is the number of processes. Notice that *non-silent* self-stabilizing leader election can be achieved using less memory, *e.g.*, the non-silent self-stabilizing leader election algorithm for unoriented ring-shaped networks given in [8] requires $O(\log\log n)$ space per process.

Self-stabilizing leader election algorithms for arbitrary connected identified networks have been proposed in the message-passing model [9–11]. First, the algorithm of Afek and Bremler [9] stabilizes in $O(n)$ rounds using $\Theta(\log n)$ bits per process. But, it assumes that the link-capacity is bounded by a value B, known by all processes. Two solutions that stabilize in $O(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network, have been proposed in [10,11]. However, both solutions assume that processes know some upper bound D on the diameter $\mathcal{D}$; and require $\Theta(\log D \log n)$ bits per process.

Several solutions are also given in the shared memory model [12,13,6,1,2,14]. The algorithm proposed by Dolev and Herman [12] is not silent, works under a *fair* daemon, and assume that all processes know a bound N on the number of processes. This solution stabilizes in $O(\mathcal{D})$ rounds using $\Theta(N \log N)$ bits per process. The algorithm of Arora and Gouda [13] works under a *weakly fair* daemon and assume the knowledge of some bound N on the number of processes. This solution stabilizes in $O(N)$ rounds using $\Theta(\log N)$ bits per process.

Datta et al. [6] propose the first self-stabilizing leader election algorithm (for arbitrary connected identified networks) proven under the distributed unfair daemon. This algorithm stabilizes in $O(n)$ rounds. However, the space complexity of this algorithm is unbounded. (More precisely, the algorithm requires each process to maintain an unbounded integer in its local memory.)

Solutions in [1,2,14] have a memory requirement which is asymptotically optimal (*i.e.* in $\Theta(\log n)$). The algorithm proposed by Kravchik and Kutten [14] assumes a synchronous daemon and the stabilization time of this latter is in $O(\mathcal{D})$ rounds. The two solutions proposed by Datta et al. in [1,2] assume a distributed unfair daemon and have a stabilization time in $O(n)$ rounds. However, despite these two algorithms stabilizing within a finite number of steps (indeed, they are proven assuming an unfair daemon), no step complexity analysis is proposed.

Contribution. We propose a silent self-stabilizing leader election algorithm for arbitrary connected and identified networks. Our solution is written in the locally shared memory model assuming a distributed unfair daemon, the weakest scheduling assumption. Our algorithm assumes no knowledge of any global parameter (*e.g.*, an upper bound on $\mathcal{D}$ or n) of the network. Like previous solutions of the literature [1,2], it is asymptotically optimal in space (*i.e.*, it can be implemented using $\Theta(\log n)$ bits per process), and it stabilizes in $\Theta(n)$ rounds in the worst case. Yet, contrary to those solutions, we show that our algorithm has a stabilization time in $\Theta(n^3)$ steps in the worst case.

For fair comparison, we have also studied the step complexity of the algorithms given in [1,2], noted here $\mathcal{DLV}$ and $\mathcal{DLV}2$, respectively. These latter are the closest to ours in terms of performance. We show that their stabilization time is not polynomial. Indeed, there is no constant α such that the stabilization time of $\mathcal{DLV}$ is in $O(n^\alpha)$ steps. More precisely, we show that fixing α to any constant greater than or equal to 4, for every $\beta \geq 2$, there exists a network of $n = 2^{\alpha-1} \times \beta$ processes in which there exists a possible execution that stabilizes in $\Omega(n^\alpha)$ steps. Similarly, for $n \geq 5$, there exists a network and a possible execution of $\mathcal{DLV}2$ that stabilizes in $\Omega(2^{\left\lfloor \frac{n-1}{4} \right\rfloor})$ steps.

Roadmap. The next section is dedicated to computational model and basic definitions. In Section 3, we propose our self-stabilizing leader election algorithm. We prove its correctness in Section 4. In the same section, we also study its stabilization time in both steps and rounds. We show that the stabilization time of the self-stabilizing leader election algorithms given in [1,2] are not polynomial in steps in Sections 5 and 6, respectively. We present some experimental results in Section 7. We conclude in Section 8.

2. Computational model

2.1. Distributed systems

We consider *distributed systems* made of n *processes*. Each process can communicate with a subset of other processes, called its *neighbors*. We denote by $\mathcal{N}_p$ the set of neighbors of process p. Communications are assumed to be bidirectional,

[1] We only consider here deterministic algorithms.

Please cite this article in press as: K. Altisen et al., Self-stabilizing leader election in polynomial steps, Inf. Comput. (2016), http://dx.doi.org/10.1016/j.ic.2016.09.002