



HAL
open science

Deterministic Leader Election in $O(D + \log n)$ Time with Messages of Size $O(1)$

Arnaud Casteigts, Yves Métivier, John Michael Robson, Akka Zemmari

► To cite this version:

Arnaud Casteigts, Yves Métivier, John Michael Robson, Akka Zemmari. Deterministic Leader Election in $O(D + \log n)$ Time with Messages of Size $O(1)$. 30th International Symposium on Distributed Computing (DISC), Sep 2016, Paris, France. pp.16-28, 10.1007/978-3-662-53426-7_2. hal-01472671

HAL Id: hal-01472671

<https://hal.science/hal-01472671>

Submitted on 21 Feb 2017

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.

Deterministic Leader Election in $O(D + \log n)$ Time with Messages of Size $O(1)$ *

A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari

Université de Bordeaux - Bordeaux INP LaBRI, UMR CNRS 5800
351 cours de la Libération, 33405 Talence, France
{acasteig, metivier, robson, zemmari}@labri.fr

Abstract. This paper presents a distributed algorithm, called STT , for electing deterministically a leader in an arbitrary network, assuming processors have unique identifiers of size $O(\log n)$, where n is the number of processors. It elects a leader in $O(D + \log n)$ rounds, where D is the diameter of the network, with messages of size $O(1)$. Thus it has a bit round complexity of $O(D + \log n)$. This substantially improves upon the best known algorithm whose bit round complexity is $O(D \log n)$. In fact, using the lower bound by Kutten et al. [13] and a result of Dinitz and Solomon [8], we show that the bit round complexity of STT is optimal (up to a constant factor), which is a step forward in understanding the interplay between time and message optimality for the election problem. Our algorithm requires no knowledge on the graph such as n or D .

1 Introduction

The election problem in a network consists of distinguishing a unique node, the leader, which can subsequently act as coordinator, initiator, and more generally performs some special role in the network (see [22] p. 262). Once a leader is established, many problems become simpler. For this reason, election algorithms are often considered as building blocks for other distributed algorithms and election, together with consensus, is probably the most studied task in distributed computing literature [7], starting with the works of Le Lann [14] and Gallager [10] in the late 70's.

A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process (or node) is in the *elected* state and all others are in the *non-elected* state. It is also required that once a process becomes elected or non-elected, it remains so for the rest of the execution. The vast body of literature on election (see [2, 15, 19, 23] and references therein) actually covers a number of different topics. They include the feasibility of deterministic election in anonymous networks, starting with the seminal paper of Angluin [1] and the key role

* A full version of this paper can be found on arXiv (<http://arxiv.org/abs/1605.01903>)

of coverings; the complexity of *deterministic* election in networks *with identifiers*; and the complexity of probabilistic election in anonymous (or sometimes identified) networks.

The present work is in the second category. We assume that each node has a unique identifier which is a positive integer of size $O(\log n)$, and the nodes exchange messages with their neighbours in synchronous rounds. The exact complexity of deterministic leader election in this setting has proven elusive for decades and even some simple questions remain open [13]. Assuming the size of messages is logarithmic (i.e. messages of size $O(\log n)$), we know since Peleg [16] that $O(D)$ rounds are sufficient to elect a leader in arbitrary networks. This was recently proven optimal by Kutten et al. [13] using a very general $\Omega(D)$ lower bound (that applies even in the probabilistic setting). Independently, Fusco and Pelc [9] showed that the time complexity of leader election is $\Omega(D + \lambda)$ where λ is the smallest depth at which some node has a unique view, called the *level of symmetry* of the network. (The view at depth t from a node is the tree of all paths of length t originating at this node.) If nodes have unique identifiers, then $\lambda = 0$, which implies the same $\Omega(D)$ bound as in [13].

Regarding message complexity, Gallager [10] presents the first election algorithm for general graphs with $O(m + n \log n)$ messages, where m is the number of edges, and a running time of $O(n \log n)$. Santoro [18] proves a matching $\Omega(m + n \log n)$ lower bound for the number of messages. A few years later, Awerbuch [3] presents an algorithm whose message complexity is again $O(m + n \log n)$, but time complexity is taken down to $O(n)$.

A number of questions remain open for election. Peleg asks in [16] whether an algorithm could be both optimal in time and in number of messages. The answer depends on the setting, but remains essentially open [13]. In the conclusion of their paper, Fusco and Pelc [9] also observe that it would be interesting to investigate other complexity measures for the leader election problem, such as *bit complexity*. This measure can be viewed as a natural extension of communication complexity (introduced by Yao [24]) to the analysis of tasks in a distributed setting.

Following [11], the bit round complexity of an algorithm \mathcal{A} is the total number of *bit rounds* it takes for \mathcal{A} to terminate, where a bit round is a round with single bit messages. This measure has become popular recently, as it captures into *a single quantity* aspects that relate both to time and to the amount of information exchanged. In this framework, the time-optimal algorithm of Peleg [16] results in a bit round complexity of $O(D \log n)$ (i.e. $O(D)$ rounds with $O(\log n)$ message size), and the

message-optimal algorithm of [3] results in a $O(n \log n)$ bit round complexity (i.e. $O(n)$ time with $O(\log n)$ message size).

In this paper, we present a bit round complexity *optimal* leader election algorithm for arbitrary synchronous networks. Our algorithm requires $O(D + \log n)$ bit rounds, and we show this is optimal by combining a lower bound from [13] and a recent communication complexity result by Dinitz and Solomon [8]. This work is thus a step forward in understanding election, and a partial answer to whether optimality can be achieved both in time and in the *amount* of information exchanged. (As opposed to measuring time on the one hand, and the number of messages *of a given size* on the other hand.) In this respect, our result illustrates the benefits of studying optimality under the unified lenses of bit complexity.

1.1 Contributions

We present an election algorithm \mathcal{STT} , having time complexity of $O(D + \log n)$ with messages of size $O(1)$, where D is the diameter of the network. Algorithm \mathcal{STT} solves the *explicit* (i.e. strong) variant of the problem defined in [13], namely, the identifier of the elected node is eventually known to all the nodes. It also fulfills requirements from [8], such as ensuring that every non-leader node knows which local link is in direction of the leader, and these nodes learn the maximal id network-wide (MaxF), as a by-product of electing this specific node in the *explicit* variant.

The architecture of our algorithm follows the same principle as many election algorithms, such as those of Gallager [10] or Peleg [16]. It relies on a competition of spanning tree constructions that works by extinction of those trees originating at nodes with lower identifiers (see Algorithm 4 in [2] and discussion therein). Eventually, a single spanning tree survives, whose root is the node with highest identifier. This node becomes elected when it detects termination (recursively from the leaves up the root). Difficulty arises from designing such algorithms with the extra constraint that only constant size messages must be used. Of course, one might simulate $O(\log n)$ -size messages in the obvious way paying $O(\log n)$ bit rounds for each message. But then, the bit round complexity would remain $O(D \log n)$. Our algorithm takes it down to $O(D + \log n)$.

For ease of exposition, we split the \mathcal{STT} algorithm into three components described below, whose execution is joint in a specific way.

1. A spreading algorithm \mathcal{S} which pipelines the maximal identifier bit-wise to each node, in a mix of battles (comparisons), conquests (progress of locally higher prefixes), and correction waves of bounded amplitude;

2. A spanning tree algorithm that executes in parallel of \mathcal{S} and whose union with \mathcal{S} is denoted \mathcal{ST} . It consists in updating the tree relations based on what neighbour brought the highest prefix so far;
3. A termination detection algorithm that executes in parallel of \mathcal{ST} and whose union with \mathcal{ST} is denoted \mathcal{STT} . This component enables the node with highest identifier (and only this one) to detect termination of the spanning tree construction rooted whose root it is.

An extra component can be added to broadcast a (constant size) termination signal from the root down the tree, once election is complete. This component is trivial and therefore not described here.

Lower Bound: Dinitz and Solomon [8] prove a lower bound (Theorem 1 below) on the leader election problem among two nodes.

Theorem 1 ([8]). *Let M be an integer such that $M \geq 2$. Let G be the graph with two nodes linked by an edge each node has a unique identifier taken from the set $Z_M = \{0, \dots, M\}$. The bit round complexity of the Leader task and of the MaxF version is exactly $2\lceil \log_2((M+2)/3.5) \rceil$.*

This theorem implies that the time complexity of an election algorithm with messages of size $O(1)$ is $\Omega(\log n)$, and thus the bit round complexity of Algorithm \mathcal{STT} is $\Omega(\log n)$.

On the other hand, the lower bound by Kutten et al. in [13], establishing that $\Omega(D)$ time is required with logarithmic size messages, obviously extends to constant size messages. Put together, these results imply that the bit complexity of leader election with messages of size $O(1)$ and identifiers of size $O(\log n)$ is $\Omega(D + \log n)$, which makes our algorithm bit-optimal (up to a constant factor).

In fact, the lower bound holds for arbitrary sizes $|id|$ of identifiers (necessarily larger than $\log n$, though, since they are unique). Likewise, the complexity of our algorithm is expressed relative to identifiers of arbitrary sizes (see Theorem 25). Hence, the bit round complexity of the election problem is in fact $\Theta(D + |id|)$. Table 1.1 summarises these elements.

Outline: After general definitions in Section 2, we present the three components of the algorithm: the spreading algorithm \mathcal{S} (Sections 3), its joint use with the spanning tree algorithm (\mathcal{ST} , Section 4), and the adjunction of termination detection (\mathcal{STT} , Section 5). We conclude in Section 6 with some remarks.

	Time	Number of messages	Message size	Bit round complexity
Awerbuch [3]	$O(n)$	$\Theta(m + n \log n)$	$O(\log n)$	$O(n \log n)$
Peleg [16]	$\Theta(D)$	$O(Dm)$	$O(\log n)$	$O(D \log n)$
This paper	$O(D + \log n)$	$O((D + \log n)m)$	$O(1)$	$\Theta(D + \log n)$

Table 1. Best known solutions in terms of time and number of messages, compared to our algorithm.

2 Model and definitions

2.1 The Network

We consider a failure-free message passing model for distributed computing. The communication model consists of a point-to-point communication network described by a connected graph $G = (V, E)$ where the nodes V represent network processes (or nodes) and the edges E represent bidirectional communication channels. Processes communicate by message passing: a process sends a message to another by depositing the message in the corresponding channel.

Let n be the size of V . We assume that each node u is identified by a unique positive integer of $O(\log n)$ bits, called identifier and denoted Id_u (in fact, Id_u denotes both the identifier and its *binary representation*). We do not assume any global knowledge on the network, not even the size or an upper bound on the size, neither do the nodes require position or distance information. Every node is equipped with a port numbering function (i.e. a bijection between the set of incident edges I_u and the integers in $[1, |I_u|]$), which allows it to identify which channel a message was received from, or must be sent to. Two nodes u and v are said to be neighbours if they can communicate through a port.

Finally, we assume the system is fully synchronous, namely, all processes start at the same time and time proceeds in synchronised rounds composed of the following three steps:

1. Send messages to (some of) the neighbours,
2. Receive messages from (some of) the neighbours,
3. Perform local computation.

The time complexity of an algorithm is the number of such rounds needed to complete the execution in the worst case.

2.2 Further definitions

The paper uses a number of definitions from graph theory and formal language theory. Although most readers may be familiar with them, we remind the most important ones. Next we define the bit round complexity.

Definitions on graphs: These definitions are selected from [17] (Chapter 8). A tree is a connected acyclic graph. A rooted tree is a tree with one distinguished node, called the root, in which all edges are implicitly directed away from the root. A spanning tree of a connected graph $G = (V, E)$ is a tree $T = (V, E')$ such that $E' \subseteq E$. A forest is an acyclic graph. A spanning forest of a graph $G = (V, E)$ is a forest whose node set is V and edge set is a subset of E . A rooted forest is a forest such that each tree of the forest is rooted. A child of a node u in a rooted tree is an immediate successor of u on a path from the root. A descendant of a node u in a rooted tree is u itself or any node that is a successor of u on a path from the root. The parent of a node u in a rooted tree is a node that is the immediate predecessor of u on a path to u from the root.

Definitions on languages: These definitions are selected from [17] (Chapter 16). Let A be an alphabet, A^* is the set of all words over A , the empty word is denoted by ϵ . If x is a non empty word over the alphabet A of length p then x can be written as the concatenation of p letters, i.e., $x = x[1]x[2] \cdots x[p]$ with each $x[i]$ in A . If $a \in A$ and i is a positive integer then a^i is the concatenation i times of the letter a . Let x and y be two words over alphabet A , x is said to be a prefix (*resp.* proper prefix) of y if there exists a word (*resp.* non-empty word) z such that $y = xz$.

Bit round complexity: The bit complexity in general may be viewed as a natural extension of communication complexity (introduced by Yao [24]) to the analysis of tasks in a distributed setting. An introduction to the area can be found in Kushilevitz and Nisan [12]. In this paper, we follow the definition from [11], that is, the bit round complexity of an algorithm \mathcal{A} is the total number of *bit rounds* it takes for \mathcal{A} to terminate, where a bit round is a synchronous round with single bit messages. This measure captures into a single quantity aspects that relate both to time and to the amount of information exchanged. Other definitions are considered in the literature, in [4–7] the bit complexity is the total number of bits sent until global termination. In [20], it is the maximum number of bits sent through a same channel. In both variants, silences may convey much information, which is why we consider the definition from [11] in terms of *round* complexity as more comprehensive.

3 A spreading algorithm

This section presents a distributed spreading algorithm using only messages of size $O(1)$ which allows each node to know the highest identifier among the set of all identifiers with a time complexity of $O(D + \log n)$, where D is the diameter of G .

3.1 Preamble

Given a node u and the binary representation Id_u of its identifier. We define $\alpha(Id_u)$ as the word

$$\alpha(Id_u) = 1^{|Id_u|}0Id_u.$$

For instance, if u has identifier 23, then $Id_u = 10111$ and $\alpha(Id_u) = 11111010111$. This encoding has the nice property that it extends the natural order $<$ of integers into a lexicographic order \prec on their α -encoding.

Remark 2. Let u and v be two nodes with identifiers Id_u and Id_v . Then:

$$Id_u < Id_v \Leftrightarrow \alpha(Id_u) \prec \alpha(Id_v).$$

As a result, the order between two identifiers Id_u and Id_v is the order induced by the first letter which differs in $\alpha(Id_u)$ and $\alpha(Id_v)$. This property is key to our algorithm, in which the spreading of identifiers progresses bitwise and comparisons occur consistently.

3.2 The algorithm \mathcal{S}

Variables: Each node can be *active* or *follower*, depending on whether it is still a candidate for becoming the leader (i.e. no higher identifier was detected so far). Each node u also has variables Y_u , Z_u and Z_u^v (one for each neighbour v of u) which are words over the alphabet $\{0, 1\}$. Y_u is a shorthand for $\alpha(Id_u)$, it is set initially and never changes afterwards. Z_u is a prefix of Y_w , for some node w (possibly u itself). It indicates the highest prefix known so far by u . On each node, this variable will eventually converge to the α -encoding of the highest identifier. Finally, for each neighbour v of u , Z_u^v is the latest value of Z_v known to u .

Initialisation: Initially every node u is *active*, all the Z_u 's are set to the empty word ϵ , and the Z_u^v 's are accordingly set to the empty word (wlog, we assume that a preliminary round made it possible for all nodes to know what neighbours they have).

Main loop: In each round, the algorithm executes the following actions.

1. update Z_u ,
2. send to all neighbours a signal indicating how Z_u was updated,
3. receive such signals from neighbours,
4. update all the Z_u^v accordingly.

The main action is the update of Z_u (step 1). It depends on the values of Z_u^v for all neighbours v and Z_u itself at the end of the previous round. This update is done according to a number of rules. For instance, as long as u remains *active* and Z_u is a proper prefix of Y_u , the update consists in appending the next bit of Y_u to Z_u . Most updates are more complex and detailed further below. The three other actions (step 2, 3, and 4 above) only serve the purpose of informing the neighbours as to how Z_u was updated, so that all Z_u^v are correctly updated. In fact, Z_u can only be updated in *seven* possible ways, each causing the sending of a particular signal among $\{\text{append0}, \text{append1}, \text{delete1}, \text{delete2}, \text{delete3}, \text{change}, \text{null}\}$, with following meaning:

- *append0* or *append1*: Z_u was updated by appending a single 0 or a single 1;
- *delete1*, *delete2*, or *delete3*: Z_u was updated by deleting one, two or three letters from the end;
- *change*: Z_u was updated by changing the last letter from 0 to 1;
- *null*: Z_u was not modified.

Each node updates its variables Z_u^v based on these signals (step 4).

Remark 3. By the end of each round, it holds that $Z_u^v = Z_v$ for any neighbour v of u . Thus from now on, Z_u^v is simply written Z_v .

We now describe the way Z_u is updated by each node u . One property that the update guarantees is that by the end of each round, if u and v are two neighbours, then Z_u and Z_v must have a common prefix followed, in each case, by at most six letters. This fact is later used for analysis.

Update of Z_u in each round: Let us denote the state of some variable X at the end of round t by X^t . For instance, we write $Z_u^0 = \epsilon$, where round 0 corresponds to initialisation. The computation of Z_u at round t results from u being active or follower, and the values of Z_u^{t-1} and Z_v^{t-1} for all neighbours v of u . It is done according to the following rules given in order of priority, i.e., $R_{1,1}$ has a higher priority than $R_{1,2}$, having itself a higher priority than R_2 , etc. Whenever a rule is applied, the subsequent rules are ignored.

- R_1 (delete). The relationship between Z_u^{t-1} and Z_v^{t-1} for any neighbour v of u may mean that a delete operation is possible. If any delete is possible, one will be carried out; if more than one is possible, the greatest will be carried out.
- $R_{1.1}$ If some Z_v^{t-1} is a proper prefix of Z_u^{t-1} and v 's last action was a *delete*, delete $\min\{|Z_u^{t-1}| - |Z_v^{t-1}|, 3\}$ letters from the end of Z_u^{t-1} ;
- $R_{1.2}$ If $Z_u^{t-1} = z0x$ with $x \neq \epsilon$ and some $Z_v^{t-1} = z1y$, delete $|x|$ letters from the end of Z_u^{t-1} ;
- R_2 (change). if $Z_u^{t-1} = z0$ and some $Z_v^{t-1} = z1y$ then change Z_u^{t-1} to $z1$ and change u 's state to *follower* if it is *active*;
- R_3 (append). if for some v , $Z_v^{t-1} = Z_u^{t-1}1x$, then Z_u^t is obtained by appending 1 to Z_u^{t-1} ;
- R_4 (append). if for some v , $Z_v^{t-1} = Z_u^{t-1}0x$, then Z_u^t is obtained by appending 0 to Z_u^{t-1} ;
- R_5 (append). if u 's state is *active* and $t < |Y_u|$, append $Y_u[t]$ to Z_u^{t-1} ;

If none of these actions apply, then Z_u remains unchanged and a *null* signal is sent. Otherwise, a signal corresponding to the resulting action is sent. We now prove some properties on Algorithm \mathcal{S} .

Lemma 4. *Whenever a node u carries out a delete operation at round t , u 's operation at round $t+1$ must be another delete operation or a change operation.*

Proof. The proof proceeds by induction on t (details in the long version).

Lemma 4 induces immediately:

Corollary 5. *A sequence of delete operations on a node u ends with a change operation on u .*

Remark 6. If a node u applies $R_{1.1}$, $R_{1.2}$, R_2 , R_3 , or R_4 then there exists a node v such that $Y_u \prec Y_v$.

Remark 7. Let u be a node. If there exists a neighbour v of u and a round t such that $|Z_u^t| < |Z_v^t|$ then u becomes *follower*.

Lemma 8. *Let u and v be two neighbours. Let t be a round number. The words Z_u^t and Z_v^t will always take one of the following forms (up to renaming of u and v) where p and w are words and a is 1 or 0:*

1. $Z_u^t = p$ and $Z_v^t = p$,
2. $Z_u^t = p$ and $Z_v^t = pw$ with $1 \leq |w| \leq 2$,
3. $Z_u^t = p0$ and $Z_v^t = p1a$,

4. $Z_u^t = p1$ and $Z_v^t = p0w$ and $|w| \leq 3$,
5. $Z_u^t = p$ and $Z_v^t = pw$ and $3 \leq |w| \leq 6$ and u has performed a delete.

Proof. The proof proceeds by examination of all possible cases (detailed proof in the long version).

The application of rule $R_{1.2}$ corresponds to item 4, thus:

Corollary 9. *If $R_{1.2}$ is applied then $0 < |x| \leq 3$ and $y = \epsilon$.*

Lemma 8 implies:

Theorem 10. *Let G be a graph of size n and diameter D such that each node u is endowed with a unique identifier Id_u which is a non negative integer. Let X be the highest identifier. After at most $|\alpha(X)| + 6D$ rounds, algorithm \mathcal{S} terminates and for each node u , $Z_u = \alpha(X)$.*

Proof. The proof proceeds by induction on the distance of a node from the highest node (detailed proof the long version).

4 A Spanning Tree Algorithm

This section explains how the computation of a spanning tree may be associated to the spreading algorithm \mathcal{S} by selecting for each node u the edge through which Z_u was modified.

Let u be a node, we add for each neighbour v , a variable $status_u^v$ whose possible values are in $\{child, parent, other\}$: it indicates the status of v for u ; initially $status_u^v = other$. The computation of the spanning tree occurs concurrently with the spreading algorithm \mathcal{S} as follows. If R_2 , R_3 , or R_4 is applied at round t relative to neighbor v , then u choses v as parent (if not already the case). Then, in addition to the signals of the spreading algorithm (indicating how Z_u was updated), u sends a signal *parent* to v and a signal *other* to its previous parent (if different from v).

After receiving signals from neighbours, in addition to the computation of the new value of Z_v for each neighbour v by Algorithm \mathcal{S} , u updates $status_u^v$. Algorithm \mathcal{ST} denotes the algorithm obtained with Rules of the spreading algorithm \mathcal{S} and actions described just above.

Remark 11. A node has no parent if and only if it is active.

Remark 12. A node has at most one parent.

The next definition introduces for each node u a word T_u that is used to prove that the graph induced by all the *parent* relations has no cycle.

Definition 13. Let u be a node, let t be a round number of the spreading algorithm \mathcal{S} ; T_u^t is equal to:

- Z_u^t if $t = 0$ or if Z_u^t has been obtained from Z_u^{t-1} thanks to R_2 or R_3 or R_4 or R_5 ;
- $Z_u^{t'}$ if Z_u^t has been obtained from Z_u^{t-1} thanks to $R_{1.1}$ or $R_{1.2}$ and $t' < t$ is the last round where $Z_u^{t'}$ has not been obtained by a delete operation.

The following lemma is a direct consequence of the definition of T_u^t , and of R_2 , R_3 and R_4 :

Lemma 14. Let t be a round number of the spreading algorithm \mathcal{S} . If v is parent of u then $T_u^t \preceq T_v^t$; furthermore if v becomes parent of u at round t then $T_u^t \prec T_v^t$ or $T_u^t = T_v^t$ and $T_u^{t-1} \prec T_v^{t-1}$.

Corollary 15. Let t be a round number. Let u_1 be a node. Let $(u_i)_{1 \leq i \leq p}$ be nodes of G such that, at round t , for $2 \leq i \leq p$ u_i is parent of u_{i-1} . Then $u_1 \neq u_p$.

Proof. Let t be a round, and let u_1 be a node. Let $(u_i)_{1 \leq i \leq p}$ be nodes of G such that, at round t , for $2 \leq i \leq p$ u_i is parent of u_{i-1} . The previous lemma implies that $(T_{u_i}^t)_{1 \leq i \leq p}$ is increasing. Considering a couple (u_j, u_{j+1}) where R_2 , or R_3 , or R_4 has been applied for the last time before t , we obtain the result. \square

Corollary 16. Let t be a round number. Let u_1 be a node. Then either u_1 is active or there exist $(u_i)_{1 \leq i \leq p}$ nodes of G such that: for $2 \leq i \leq p$ u_i is parent of u_{i-1} and u_p is active.

Definition 17. We denote by $\mathcal{ST}(G)$ the subgraph of $G = (V, E)$ having V as node set and there is an edge between the node u and the node v if u is the parent of v or v is the parent of u when algorithm \mathcal{ST} terminates.

When Algorithm \mathcal{ST} terminates there is exactly one *active* node: the node with highest identifier. Now, from Remark 12 and Corollary 16:

Proposition 18. Let G be a connected graph such that each node has a unique identifier. Let u be the node with the highest identifier. When algorithm \mathcal{ST} terminates, the graph $\mathcal{ST}(G)$ is a spanning tree of G .

5 Termination Detection of Algorithm \mathcal{ST}

This section presents some actions which, added to algorithm \mathcal{ST} , enable the node with the highest identifier to detect termination of algorithm

\mathcal{ST} ; furthermore, as it is the only one, when it detects the termination it becomes elected. Our solution is a bitwise adaptation of the propagation process with feedback introduced in [21] and further formalised and studied in Chapter 6 and 7 of [23].

Definition 19. *Let v be a node. Let t be a round number of the spreading algorithm. The variable Z_v^t is said to be well-formed if there exists an identifier Id such that $Z_v^t = \alpha(Id)$.*

Each node v is equipped with a boolean variable $Term_v$ which is *true* iff v and all of its subtree have terminated. Whenever a rule of the spreading algorithm is applied to node v , the variable $Term_v$ is set to *false*, and a signal is sent to its neighbours to indicate that $Term_v = false$. Indeed, this variable can be updated several times for a same node before stabilizing to *true*.

We describe an extra rule to be added to the \mathcal{ST} algorithm in order to allow the node with highest identifier to learn that it is so by detecting termination of the spanning tree algorithm. This rule is considered *after* those of algorithm \mathcal{ST} in each round. Let us denote by N_v the set of neighbours of v , and by $Ch_v \subseteq N_v$ those which are v 's children. Also recall that we omit the round number in the expression on variables when it is non ambiguous.

The rule: Given a node v , if (v is follower) and ($Term_v = false$) and (Z_v is well-formed) and ($\forall w \in N_v Z_w = Z_v$) and ($\forall w \in Ch_v Term_w = true$) then $Term_v := true$. Furthermore v sends to his parent a signal indicating that $Term_v = true$.

We denote by \mathcal{STT} the algorithm obtained by putting together the rules of Algorithm \mathcal{ST} and this extra rule for termination detection.

Remark 20. Let v be a node, if $Term_v = true$ then Z_v has the same value it had when $Term_v$ became *true* the last time.

Remark 21. If $Ch_v = \emptyset$, i.e., v is a leaf, and Z_v is well-formed and for each neighbour w of v $Z_w = Z_v$ then v sets $Term_v$ to *true* right away (and v sends to his parent a signal indicating that $Term_v = true$).

Remark 22. Let u be the node with highest identifier. Let v be a node. If $Z_v = \alpha(Id_u)$ then Z_v will never change.

Theorem 10 and Proposition 18 imply:

Proposition 23. *Let G be a graph such that each node has a unique (integer) identifier. Algorithm STT terminates. Furthermore, if the node u has the highest identifier then, after a run of algorithm STT , for each neighbour v of u $Z_v = \alpha(Id_u)$ and $Term_v = true$ and the node u receives from each node v in Ch_u the signal indicating that $Term_v = true$.*

The next proposition established that only the node with highest identifier can receive a termination signal from all neighbors.

Proposition 24. *Let G be a graph such that each node has a unique identifier. Let v be a node which has not the highest identifier and such that $Z_v = \alpha(Id_v)$ and for each neighbour w of v $Z_w = Z_v$. Then there exists a neighbour v' of v such that $Term_{v'} = false$.*

Proof. The proof relies on transitive relations between $Term_v$ values within the tree (detailed proof in the long version).

If the node u with highest identifier, becomes *elected* as soon as, for each neighbour v of u , $Z_v = \alpha(Id_u)$ and $Term_v = true$ and it receives from each child v the signal indicating that $Term_v = true$ we deduce:

Theorem 25. *Let G be a graph such that each node has a unique identifier which is an integer. Let u be the node with the highest identifier. There exists an election algorithm for G with messages of size $O(1)$ which terminates after at most $|\alpha(Id_u)| + 6D$ rounds.*

6 Conclusion

Concerning deterministic election algorithms with identifiers, we may consider three complexity measures: time complexity, message complexity, and bit (round) complexity. Santoro [18] proved that $\Omega(|E| + n \log n)$ is a lower bound for the number of messages and Awerbuch [3] presented an algorithm that matches this bound. Kutten et al. [13] shows that concerning the time complexity $\Omega(D)$ is a lower bound and [16] implies that $O(D)$ is a tight upper bound. For bit (round) complexity, we deduced from [13] and [8] that $\Omega(D + \log n)$ is a lower bound and we presented an algorithm that matches this bound with a running time of $O(D + \log n)$ bit rounds. Our algorithm requires no knowledge on the graph such as the size or the diameter.

References

1. D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on Theory of Computing*, pages 82–93, 1980.

2. H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004.
3. B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In *Proc. of 19th Symp. on Theory of Computing, 1987, New York, USA*, pages 230–240, 1987.
4. A. Bar-Noy, J. Naor, and M. Naor. One-bit algorithms. *Distributed Computing*, 4:3–8, 1990.
5. H. L. Bodlaender, S. Moran, and M. K. Warmuth. The distributed bit complexity of the ring: from the anonymous case to the non-anonymous case. *Inf. and comput.*, 114(2):34–50, 1994.
6. H. L. Bodlaender and G. Tel. Bit-optimal election in synchronous rings. *Inf. Process. Lett.*, 36(1):53–56, 1990.
7. Y. Dinitz, S. Moran, and S. Rajsbaum. Bit complexity of breaking and achieving symmetry in chains and rings. *Journal of the ACM*, 55(1), 2008.
8. Y. Dinitz and N. Solomon. Two absolute bounds for distributed bit complexity. *Theor. Comput. Sci.*, 384(2-3):168–183, 2007.
9. E. G. Fusco and A. Pelc. Knowledge, level of symmetry, and time of leader election. *Distributed Computing*, 28(4):221–232, 2015.
10. R. G. Gallager. Finding a leader in a network with $o(e+n \log n)$ messages. *Technical Report Internal Memo., M.I.T., Cambridge, MA*, 1979.
11. K. Kothapalli, M. Onus, C. Scheideler, and C. Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. In *20th Int. Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece*. IEEE, 2006.
12. E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1999.
13. S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, 2015.
14. G. LeLann. Distributed systems: Towards a formal approach. In B. Gilchrist, editor, *Information processing'77*, pages 155–160. North-Holland, 1977.
15. N. A. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.
16. D. Peleg. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.*, 8(1):96–99, 1990.
17. K. H. Rosen, editor. *Handbook of discrete and combinatorial mathematics*. CRC Press, 2000.
18. N. Santoro. On the message complexity of distributed problems. *International Journal of Parallel Programming*, 13(3):131–147, 1984.
19. N. Santoro. *Design and analysis of distributed algorithm*. Wiley, 2007.
20. J. Schneider and R. Wattenhofer. Trading bit, message, and time complexity of distributed algorithms. In *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, pages 51–65, 2011.
21. Adrian Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, 29(1):23–34, 1983.
22. A. Tanenbaum and M. van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.
23. G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.
24. A. C. Yao. Some complexity questions related to distributed computing. In *Proc. of 11th Symp. on Theory of computing (STOC)*, pages 209–213. ACM Press, 1979.