A Self-Stabilizing Leader Election Algorithm for Tree Graphs[*]

Gheorghe Antonoiu and Pradip K Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Technical Report CS-96-115

[*]To appear in *Journal of Parallel and Distributed Computing*

A Self-Stabilizing Leader Election Algorithm for Tree Graphs[*]

Gheorghe Antonoiu and Pradip K Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Abstract

We propose a self stabilizing algorithm (protocol) for leader election in a tree graph. We show the correctness of the proposed algorithm by using a new technique involving induction.

1 Introduction

In a distributed system the computing elements or nodes exchange information only by message passing. Every node has a set of local variables whose contents specify the local state of the node. The state of the entire system, called the *global state*, is the union of the local states of all the nodes in the system. Each node is allowed to have only a partial view of the global state, and this depends on the connectivity of the system and the propagation delay of different messages. Yet, the objective in a distributed system is to arrive at a desirable global final state (legitimate state), defined by some invariance relation on the global state. Systems that reach the legitimate state starting from any arbitrary (possibly illegitimate) state in a finite number of steps are called self-stabilizing systems [Dij74, Dij86]. This kind of property is highly desirable for fault tolerance [Lam84] in distributed systems, since without having a global memory, global synchronization is achieved in finite time without any intervention by any external agency and thus the system can correct itself automatically from spurious perturbations or failures. Few such algorithms have

[*]To appear in *Journal of Parallel and Distributed Computing*

recently appeared in the literature [GH90, BGW89, ADG92, CD94, FD94, CS94]; a good survey of self-stabilizing algorithms can be found in [Sch93].

Our purpose in the present paper is to propose a self-stabilizing leader election protocol for tree graphs. Thus, a global state, where an arbitrary node can be unambiguously elected *leader* as identified by some characteristics (unique from all other nodes), is a legitimate state. Self stabilizing leader election protocols for uniform rings of primal size has been proposed in [Hua93], while it has been shown in [BP89] that self stabilizing protocol for leader election cannot exist for a ring of composite size. We first show that it may not be possible to design a self stabilizing leader election protocol for a tree that can elect a leaf node to be the leader; but, it is possible to get a self stabilizing protocol to elect an arbitrary internal node as the leader if a node is allowed to test if it is a leaf or internal node and accordingly take an appropriate action. This is quite natural for a tree graph while there is no apparent way to do this in a ring; note that in all existing self-stabilizing algorithms each node can get the information about the states of its neighbor nodes and hence does effectively know the number of its neighbors. In this sense, our protocol does have a single uniform rule for each node of the tree and ecah node uses the information about its own state and its neighbor states.

Each node, in our algorithm, has a level variable ranging over the domain $\{0, 1, 2, \cdots, C\}$ where C is a positive integer greater than or equal to the number of the nodes in the tree. The level variables initially have arbitrary values; the algorithm changes the levels in such a way that the system reaches, in finite time, a state where a unique but arbitrary internal node has a level that is strictly greater than that of all other nodes; this internal node can then be treated as the leader.

One useful startegy to prove the correctness of self stabilizing algorithms is to use bounded monotonically decreasing functions defined on global system states [Kes88]; most existing self-stabilizing algorithms are proved to be correct by defining a bounded function that is shown to decrease monotonically at every step [Hua93]. We do not use this technique; instead we develop a new proof technique using induction on the number of nodes in the tree; this may prove useful in proving the correctness of other self stabilizing protocols. Most self-stabilizing algorithms assume that there is a *central daemon* [Dij74] that decides which of the privileged nodes makes a move. In other words, the central daemon serializes the moves made by the privileged nodes, but the order in which the privileged nodes are chosen to make their moves is not known a priori. However, the presence of such a daemon is against the fundamental idea of a distributed system. Our algorithm does uses a much weaker assumption; when multiple nodes are privileged, an arbitrary subset of these nodes (at least one) can make a move as long as the active nodes are not adjacent. Our algorithm does not assume any order or any specific rule in which the set of active nodes is chosen at any point of time.

2 The Leader Election Algorithm

Consider an arbitrary tree graph $T = (V, E)$. We use the following notations:

- *n:* The number of nodes in the tree.
- *C:* An integer constant such that $C \geq n$.
- $\mathcal{N}(i)$*:* Set of the neighbors of node i.
- *L(i):* Level of node i.
- *P(i):* Predecessor pointer of node i, pointing to one of the nodes in $\mathcal{N}(i)$.

Thus each node i maintains two data structures $L(i)$ and $P(i)$ which can have arbitrary initial values. Note that $0 \leq L(i) \leq C$; we do not need to consider level values beyond that (even after perturbation) as we can always assume each processor is capable of doing a modulo $(C + 1)$ operation and always keeps the remainder $(\bmod\ (C + 1))$ as its level value. Also, we do not assume the nodes are assigned unique identification numbers. Each node maintains an ordered list of its neighbors and the predecessor pointer $P(i)$ at node i is set to point to one of its neighbors or null.

Definition 1 *A node i is called* **leaf node** *iff* $|\mathcal{N}(i)| = 1$. *A node i is called* **internal node** *iff* $|\mathcal{N}(i)| > 1$.

Definition 2 *For each node i we define the following sets:* $N_L(i) = \{j \in \mathcal{N}(i) | L(j) < L(i)\}$, $N_E(i) = \{j \in \mathcal{N}(i) | L(j) = L(i)\}$, *and* $N_G(i) = \{j \in \mathcal{N}(i) | L(j) > L(i)\}$. *For each node i we also define two integers:* $m(i) = max\{L(k) | k \in N_L(i)\}$ *and* $M(i) = max\{L(k) | k \in N_E(i) \cup N_G(i)\}$.

We make the following immediate observations:

- In any given system state, the sets $N_L(i)$, $N_E(i)$, and $N_G(i)$ for each node i in the tree give the subset of neighbors of node i whose levels are respectively less than, equal to and greater than that of node i. Of course, one or more of the sets for a node i may be null, but all three cannot be simultaneously null (T is a connected graph).

- In any given system state, the sets $N_L(i)$, $N_E(i)$, and $N_G(i)$ for each node i are pairwise mutually disjoint.

- Since each node looks at its own state and the states of its neighbors, each node can compute these three sets in any system state.

- If $|N_L(i)| = 0$, then $m(i)$ is not defined; similarly, if $|N_E(i) \cup N_G(i)| = 0$, then $M(i)$ is not defined.

Definition 3 *In any given system state, a node i is called* **maximal** *iff* $|N_G(i)| = 0$ *and is called* **strictly maximal** *iff* $|N_G(i) \cup N_E(i)| = 0$. *Similarly, a node i is called* **minimal** *iff* $|N_L(i)| = 0$ *and is called* **strictly minimal** *iff* $|N_L(i) \cup N_E(i)| = 0$.

The concept of the legitimate state depends on what we want to maintain. Since our objective in the present paper is leader election, any possible system state, where an arbitrary node can be unambiguously elected *leader* as identified by some characteristics (unique from all other nodes), may be called a legitimate state. Note that this requirement does not say anything about the characteristics of the other nodes as long as one node can be singled out based on some characteristics. The second requirement is that the rest of the nodes in the tree T are aware of the identification of the leader node. Any other system state is an illegitimate state.

The purpose of a self-stabilizing algorithm is to bring back the stable state once the system is in any possible illegitimate state by any perturbation. The basic idea is whenever the system is in an illegitimate state at least one of the nodes should be able to recognize some local inconsistency and should take some corrective action. Any action taken by a node in an illegitimate state depends only on its own state and the states of its neighboring nodes (each node has only a partial view of the global system state). As noted earlier, electing a leader does not need that each node in the tree T is assigned an unique identification; only the leader node needs to have a unique identification. But, since each node needs to know the leader, an actual algorithm may restrict the characteristics of the other nodes in some way. First, we state the following theorem.

Theorem 1 *There can be no self-stabilizing algorithm that assigns distinct numbers to the leaf nodes of an arbitrary tree graph.*

Proof : Consider a tree of 3 nodes: an internal node r and two leaf nodes a and b. Each node x maintains some local data structure $S(x)$. When a node x takes an action, the new value of $S(x)$ depends only on its own state and those of its neighbors. Consider an illegitimate state with identical initial data structures for nodes a and b; this has to be an illegitimate state for a self-stabilizing algorithm that would assign distinct numbers to leaf nodes. So, either the internal node is privileged, or the nodes a, b are privileged, or both. If the internal node takes an action, the new state will also have have the property $S(a) = S(b)$. If a, b are privileged and one of them takes an action, the other one will remain privileged, since the state of the internal node – its only neighbor – didn't change. Therefore, the node which didn't take action can take action and the new state will have $S(a) = S(b)$. Hence, repeating the argument, there is an infinite sequence of moves such that each state in the sequence has at least one privileged node. □

Corollary 1 *There is no general sef-stabilizing algorithm that assigns distinct numbers to the nodes of a tree.*

Theorem 2 *There is no general sef-stabilizing algorithm that can choose a leaf node as a leader.*

Proof : Since a system state with identical data structures for leaf nodes is also an illegitimate state for any algorithm that elects a leaf node as leader, the proof is similar to that of the previous theorem. □

Remark 1 *The theorem does not exclude the existence of such an algorithm for some particular classes of trees.*

Thus, the proposed algorithm would strive to elect an arbitrary internal node as the leader. We restrict ourselves to the first requirement of leader election and then later show that once the legitimate state is reached, any non leader node can reach the leader node using the predecessor pointers. Our algorithm has a single uniform rule for all the nodes in the graph. Each node looks at its own state and the states of its neighbors and takes action by changing its own level. Thus, the global system state is defined by the union of the level values at each node; the state space, although very large, is finite. We introduce a predicate Ψ_i for any node i;

$$\Psi_i : L(i) > m(i) + 1,$$

Note that Ψ_i is defined only when $m(i)$ is defined for the node i. Ψ_i is true for a node i iff the difference of $L(i)$ and the maximum of the level of the nodes in $N_L(i)$ is 2 or more. Note that $\neg\Psi_i \equiv L(i) = m(i)+1$, since $L(i)$ and $m(i)$ are integers and by definition $m(i) < L(i)$. We can now state the algorithm as a single rule for each node in the graph. The rule at node i is as follows:

$$(\mathbf{R}) \quad \begin{cases} \textbf{if } |\mathcal{N}(i)| = 1 \wedge L(i) \neq 0 \textbf{ then } L(i) = 0; \\ \textbf{else if } \Psi_i \wedge |N_G(i) \cup N_E(i)| \leq 1 \textbf{ then } L(i) = m(i) + 1; \\ \textbf{else if } L(i) < C \wedge [|N_G(i) \cup N_E(i)| \geq 2 \vee \{|N_G(i)| = 0 \wedge |N_E(i)| = 1 \wedge \neg\Psi_i\}] \\ \qquad\qquad \textbf{then } L(i) = \min\{M(i) + 1, C\}; \end{cases}$$

Note: By construction of the algorithm whenever Ψ_i needs be evaluated, it is always defined; Ψ_i needs be evaluated only for an internal node j (with $|\mathcal{N}(j)| > 1$) such that $|N_G(j) \cup N_E(j)| \leq 1$ (thus, $|N_L(j)| \geq 1$).

Definition 4 *When an internal node x executes $L(x) = m(x) + 1$ (second clause of the algorithm), level of node x decreases after this move (since Ψ_x is true); we call this a* **D-move***. When an internal node x executes $L(x) = M(x) + 1$, level of node x increases after this move; we call this an* **I-move***.*

Remark 2 *A leaf node x is privileged iff $L(x) \neq 0$; a leaf node may be privileged in an illegitimate state, but once it takes action it becomes un privileged and can never be privileged again.*

Remark 3 *An internal node x with $L(x) < C$ is privileged when $|N_E(x)| > 0 \vee |N_G(x)| > 1 \vee \Psi_x$; an internal node x with $L(x) = C$ is privileged when $\Psi_x \wedge |N_E(x)| \leq 1$ (note that for a node x with $L(x) = C$, $N_G(x) = \emptyset$). Note that when an internal node is privileged, it can take either a D-move or an I-move, but not both. If it makes a D-move, it becomes un privileged after the move but may be privileged subsequently due to moves by its neighbors. If it makes an I-move to make its level C, it may be still privileged to make a D-move only.*

Lemma 1 *In any system state, if all nodes are un privileged, $\forall x \in T$, $L(x) < C$.*

Proof : Suppose otherwise. If a node x has $L(x) = C$ and is un privileged, then either $N_E(x) > 1$, or $\neg\Psi_x$ or both. Consider the subset of nodes in T with level C. This subset forms a subtree of T (consider two nodes $x, y \in T$ with levels C; since T is a tree, there exists a unique path between x and y and each node on this path must have a level C since they are all un privileged). Hence, there must exist at least one node x such that $N_E(x) \not> 1$ and hence $\neg\Psi_x \equiv L(x) = m(x) + 1$; thus there exists a node y such that $L(y) = C - 1$. Since the node y is unprivileged, $L(y) = m(y) + 1$, i.e., there exists a node z such that $L(z) = C - 2$. Repeating this argument, we must have nodes with all the distinct level values from C to 0. This is a contradiction since the total number of nodes is $n \leq C$, hence the proof. □

At this point, we can have a precise definition of the legitimate (stable) state. A legitimate or stable state is a global system state when no node is privileged. Specification of the set of legitimate states depends on the algorithm and can be obtained by complementing the predicates in the algorithm.

Definition 5 *The system is in a legitimate state iff (i)* $\forall x \in T$ *s.t.* x *is a leaf node,* $L(x) = 0$*, (ii)* $\forall x \in T$ *s.t.* x *is an internal node,* $L(x) < C \wedge |N_E(x)| = 0 \wedge |N_G(x)| \leq 1 \wedge L(x) = m(x) + 1$.

Remark 4 *In any illegitimate state, at least one node in the tree* T *is privileged. This directly follows from Remarks 2 and 3 and Definition 5.*

Lemma 2 *In an illegitimate state, if a node* x *with* $L(x) < C$ *is maximal but not strictly maximal, then it must be privileged; a strictly maximal node* x *is privileged iff* Ψ_x *is true.*

Proof : A node x with $L(x) < C$, which is maximal but not strictly maximal, must have $|N_E(x)| > 0$; a strictly maximal node x has $|N_G(x) \cup N_E(x)| = 0$. The rest follows from Remark 3. □

Lemma 3 *For a tree* T *with* n *nodes,* $n \geq 3$*, in a legitimate (stable) state (i.e., no node is privileged), there exists exactly one maximal node which is also strictly maximal. The level of this strictly maximal node is less than* n*.*

Proof : Since no node is privileged, $\forall x \in T$, $L(x) < C$ (Lemma 1). Thus, any un privileged maximal node must be strictly maximal (Lemma 2). Consider the node x such that $L(x) = max\{L(k) \mid k \in V\}$. Since this node x is maximal and un privileged, it is strictly maximal. Suppose, another strictly maximal node y exists. Then, there is a path between x and y via at least one other node (both x and y are strictly maximal). Let be z the node with the minimum level on this path. Since this node z has at least two neighbors with levels greater than or equal to its own level, node z is privileged, which is a contradiction. Using an argument similar to that used in Lemma 1, there is a path from node x to a leaf such that on that path the levels of the nodes are $L(x),\ L(x) - 1,\ \ldots,\ 1,\ 0$. Since there are at most n nodes in this path, $L(x) < n$. □

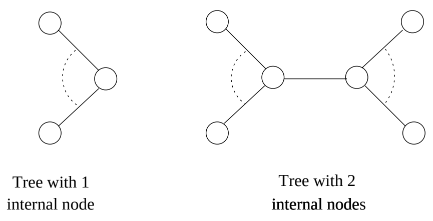


Figure 1: Star like Trees

Corollary 2 *In any legitimate state, the unique strictly maximal node has a level that is strictly greater than the level of any other node.*

Lemma 4 *In any legitimate state, the unique strictly maximal node is the elected leader and each node can reach the leader.*

Proof : The leader node x has $|N_G(x)| = 0$ and it can set its predecessor pointer $P(x) = NULL.$ Any other node $y \in T$ has $|N_G(y)| = 1$ and hence can set its predecessor pointer $P(y)$ to point to the unique node in $N_G(y)$. □

Definition 6 *An internal node in a tree T is called a* **type I** *node if it has at least two internal node neighbors in T; otherwise it is called a* **type II** *node. Thus, the vertex set of T can be partitioned as $T_1 \cup T_2 \cup T_3$, where T_1 is the set of type I internal nodes, T_2 is the set of type II internal nodes and T_3 is the set of leaf nodes of the tree T.*

Remark 5 *For a given tree T, if $T_1 = \emptyset$, then the tree T must be either a star or two stars connected by an edge; see Figure 1. For any other tree T, the set T_1 is not null.*

Definition 7 *For a given tree T and a given system state, let $\ell_{min} = \min\{L(x)|x \in T_1 \cup T_2\}$, the minimum level of internal nodes (both type I and type II). Let $L_{min} = \{x \in T_1 \cup T_2 | L(x) = \ell_{min}\}$.*

Lemma 5 *In any illegitimate state, provided no leaf node makes any move, if a type II internal node y ($y \in T_2$) makes a D-move such that $L(y)$ becomes less than $L(z)$ after the move, where $L(z)$ is the level of node y's unique type I neighbor z, node y becomes permanently un privileged.*

Proof : When node y makes a D-move, $|N_G(y) \cup N_E(y)| \leq 1$; and $L(z)$ must be equal to or greater than $L(y)$ in order that new level of node y (after the move) becomes less than $L(z)$. Thus, after the move, level of node y is one greater than the maximum of the levels of its leaf neighbors and is less than that of node z. Leaf nodes do not make any move and for any subsequent move of the node z, its level will always be greater than that of node y (since $y \in N_L(z)$); thus the node y becomes permanently un privileged. $\Box$

Lemma 6 *Starting from any illegitimate state, provided no leaf node makes any move, one of the following will occur in finite time:*

- *If $|L_{min}| > 1$, $|L_{min}|$ will decrease.*
- *If $|L_{min}| = 1$, ℓ_{min} will increase.*
- *At least one internal node will become permanently un privileged.*

Proof : We consider two cases:

Case 1: $\ell_{min} = C$. In this case, $L_{min} = T_1 \cup T_2$. No nodes in T_1 is privileged since it has at least two neighbors with level C. Any node y in T_2, if it is privileged, can make only a D-move. Any such node y has leaf node neighbors and exactly one T_1 neighbor, say z ($L(z) = C$). After the D-move, the new level of node y will be $m(y) + 1 < C$; node y is now un privileged and remain so permanently since leaf nodes do not make any move (Lemma 5).

Case 2: $\ell_{min} < C$. Consider a node $y \in T_1 \cup T_2 - L_{min}$, if this set is not empty. $L(y) > \ell_{min}$. If node y makes an I-move, the new level of node y is still greater than ℓ_{min} and hence $|L_{min}|$ does not increase. For a D-move of node y, we consider two subcases.

Subcase 1: If $y \in T_1$, the new level of y would be $m(y) + 1$ ($m(y) \geq \ell_{min}$) and hence $|L_{min}|$ does not increase.

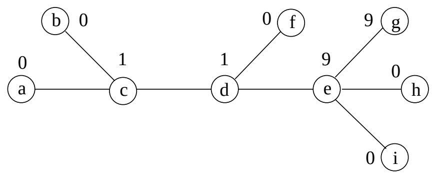
Subcase 2: If $y \in T_2$, node y has leaf node neighbors and exactly one T_1 neighbor, say z (note $L(z) \geq \ell_{min}$). If the new level of node y (after the D-move) is greater than $L(z)$, $|L_{min}|$ does not increase; otherwise, if this new level of node y is smaller than $L(z)$, node y becomes permanently un privileged (Lemma 5).

Thus, for any move made by a node $y \in T_1 \cup T_2 - L_{min}$, either the theorem is proved or $|L_{min}|$ does not increase. Now, consider a node $y \in L_{min}$. If node y does not make a move in finite time, it is permanently un privileged and the theorem is proved. Otherwise, consider again two subcases. If $y \in T_1$, y has at least two internal node neighbors, say y_1 and y_2. We have, $L(y_1) \geq \ell_{min}$, $L(y_2) \geq \ell_{min}$, and $L(y) = \ell_{min}$; thus node y must make an I-move. Hence, if $|L_{min}| > 1$, $|L_{min}|$ will decrease, or ℓ_{min} will increase. If $y \in T_2$ (node y has leaf node neighbors and exactly one T_1 neighbor, say z), y can make either an I-move or a D-move. In case of an I-move, if $|L_{min}| > 1$, $|L_{min}|$ will decrease, or ℓ_{min} will increase. In case of a D-move, level of node y will be less than that of node z and hence node y is permanently un privileged.
$\square$

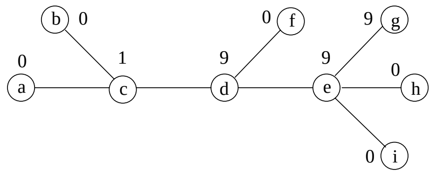
Remark 6 *Note that the argument holds as long as adjacent nodes do not make concurrent moves.*

Lemma 7 *For a tree T with n nodes, starting from any illegitimate state, there is no infinite sequence of moves made only by internal nodes.*

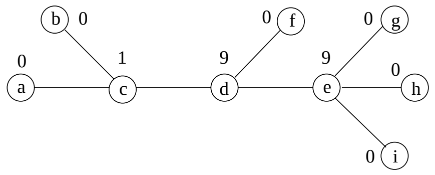
Proof : We prove by induction. For $n = 2$, the claim holds trivially since there is no internal node. For $n = 3$, there is only one possible tree structure with one internal node; the internal node is either un privileged, or becomes un privileged after its first move. Assume the lemma is true for any tree with $(n-1)$ nodes and for any $C \geq (n-1)$. Consider a tree with n nodes and $C \geq n$; assume that internal nodes make an infinite sequence of moves; we will reach a contradiction. By repeated application of Lemma 6, ℓ_{min} will gradually increase (no node can be rendered permanently un privileged because of our induction hypothesis) and in finitely many moves, ℓ_{min} will be equal to C and L_{min} will include all the internal nodes in the tree. At this point, any internal node is either un privileged or can make at most one D-move and become permanently un privileged. Thus we reach a contradiction and hence the proof.
$\square$



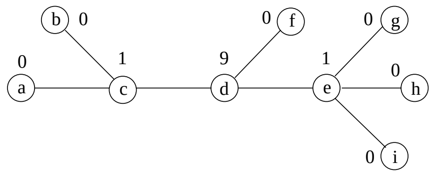
(i) $PV = \{c, d, e, g\},\ \ AV = \{d\}$



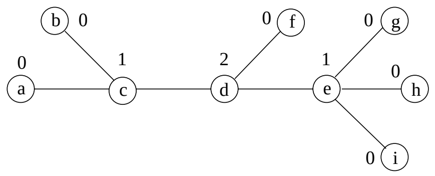
(ii) $PV = \{d, g\},\ \ AV = \{g\}$



(iii) $PV = \{d, e\},\ \ AV = \{e\}$



(iv) $PV = \{d\},\ \ AV = \{d\}$



(v) $PV = \emptyset,\ \ AV = \emptyset$

Figure 2: The Execution of the Algorithm

Lemma 8 *For a tree T with n nodes, starting with any illegitimate state, all nodes are un privileged in finitely many moves.*

Proof : It follows from Lemma 7 and the fact that any leaf node can make at most one move. □

We can now state the following theorem about the termination of the algorithm in finite time.

Theorem 3 *For a tree T with n nodes, starting with any illegitimate state, stable state is reached by the algorithm in finite time.*

Example: Figure 2 illustrates the execution of the algorithm on an example graph from an arbitrary initial state. The tree in our example has 9 vertices, $\{a, b, c, d, e, f, g, h, i, j\}$. Each node in the figures is labeled with its name and its level. The set PV denotes the set of privileged nodes and the set AV denotes the set of active nodes. It is to be noted that in a given state multiple nodes may be privileged and more than one node may be active as long as active nodes are not adjacent; we have arbitrarily chosen a subset to be active in our example. Figure 2(i) is the initial state and the Figure 2(v) is the final legitimate state. It is to be noted that there are many other possible sequences of moves that will bring the state back to a stable state starting from the same initial illegitimate state.

3 Conclusion

We have proposed a self stabilizing algorithm for leader election in a tree graph. When the algorithm terminates (in finite time), there is an unique node with a level value that is strictly greater than the levels of all other nodes; this is the elected leader node and each of the rest of the nodes has a unique way to reach that leader. The nodes in teh tree are treated uniformly in the sense that each node executes a single uniform rule. Ecah node has only a partial view of the global state; it knows of its own state and the states of its neighbors. Starting from any illegitimate state, the algorithm can elect an arbitrary internal node to be the new leader; but no leaf node will ever be selected as the leader of the tree (a leaf node in a tree is a node with eaxctly one neighbor). We have shown that it may not be possible to design a self stabilizing protocol that can elect a leaf node to be the leader. To do that, it may be interesting to look at probabilistic self-stabilizing algorithms.

References

[ADG92] A. Arora, S. Dolev, and M. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1(1):11–18, 1992.

[BGW89] G. M. Brown, M. G. Gouda, and C. L. Wu. Token systems that self-stabilize. *IEEE Trans. Comput.*, 38(6):845–852, June 1989.

[BP89] J. Burns and J. Pachl. Uniform stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11(2):330–344, 1989.

[CD94] Z. Collin and S. Dolev. Self-stabilizing depth-first search. *Information Processing Letters*, 49:297–301, 1994.

[CS94] S. Chandrasekar and P. K. Srimani. A self-stabilizing distributed algorithm for all-pairs shortest path problem. *Parallel Algorithms and Applications*, 4(1&2):125–137, 1994.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[Dij86] E. W. Dijkstra. A belated proof of self-stabilization. *J. of Distributed Computing*, 1(1):5–6, 1986.

[FD94] M. Flatebo and A. K. Datta. Two state self stabilizing algorithms for token rings. *IEEE Transactions on Software Engineering*, 20(6):500–504, June 1994.

[GH90] M. Gouda and T. Herman. Stabilizing unison. *Inf. Processing Letters*, 35(4):171–175, 1990.

[Hua93] S. T. Huang. Leader election in uniform rings. *ACM Transactions on Programming Languages and Systems*, 15(3):563–573, July 1993.

[Kes88] J. L. W. Kessels. An exercise in proving self-stabilization with a variant function. *Inf. Processing Letters*, 29(2):39–42, 1988.

[Lam84] L. Lamport. Solved problems, unsolved problems, and non-problems in concurrency. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, 1984.

[Sch93] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, March 1993.