RI 02009 3 April 2002 Computer Science

# IBM Research Report

## Self-stabilizing maxima finding on general graphs

**Vinayaka Pandit**
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

**Prof. D M Dhamdhere**
Dept CS & E, IIT-Bombay
Powai, Mumbai

# Self-stabilizing maxima finding on general graphs

Vinayaka Pandit* D M Dhamdhere†

April 3, 2002

**Abstract**

Self-stabilization is an approach for fault tolerance in presence of transient faults in distributed systems. Leader election in distributed systems is of interest because a leader is used for many key tasks like co-ordination, check-pointing, and acting as servers. In this paper, we consider the problem of electing a leader in networks of any topology in a self-stabilizing manner. We do not make any assumptions about the nature of transient faults, and effectively deal with faults like corrupted messages, transmission errors, and fake ids. The robustness of our protocol is demonstrated by the fact that we do not make assumptions about knowledge of diameter of the network, or channel properties. Our protocol also constructs a minimum depth spanning tree of the network rooted at the leader. This is a desirable property in applications where communication between a node and the leader takes place along the path in the spanning tree.

## 1 Introduction

Decentralized maxima finding has a number of applications in distributed systems. Leader election, and distributed resource allocation are obvious examples of such applications. In leader election, a leader may be elected based on an attribute like an identifier. In resource allocation, the notion of maxima changes dynamically. The leader in a distributed system can be used to co-ordinate various important tasks like synchronization, collecting check points, and acting as a server. Typically, the leader is elected based on a notion of maximality with respect to a static attribute like an identifier, or a dynamic attribute like available resources with a node. The need for fault-tolerance in these protocols is obvious due to the important role played by a leader in the above mentioned tasks. While fault-tolerance in presence failure models like crash of a node, or a malfunctioning node have been well studied, transient faults are relatively less explored. A transient fault is a fault which changes the state of a system, but not its behavior [8]. Typically, these are the most frequent faults in a large distributed system encompassing faults like corrupted data packets, duplicate packets, corrupted buffer space, and many others. Self-stabilization is a unified fault tolerance design technique for transient faults [8, 5]. In this paper, we deal with a transient failure model. We use the terms *maxima finding* and *leader election* synonymously.

The notion of self-stabilization was introduced by Dijkstra [5]. He defined self-stabilization as a property by which, irrespective of the initial state of the system, it is guaranteed to arrive at a legitimate state in a finite number of steps, and remaining in a legitimate state after reaching a legitimate state for the first time. Formally, a system $S$ is said to be self-stabilizing with respect to a property $P$ if the system satisfies the following conditions under the execution of the algorithm.

*pvinayak@in.ibm.com
†dmd@cse.iitb.ernet.in

They are (i) *Closure:* $P$ is closed under execution of $S$ which implies that once $P$ is established in $S$, it cannot be falsified, and (ii) *Convergence:* Starting from an arbitrary global state, $S$ is guaranteed to enter a state satisfying $P$ in finite number of steps [8]. Two of the most important properties of self-stabilizing systems, especially in large distributed systems are, (i) they need not initialized, and (ii) they can automatically recover from transient faults. The size, and the topology of the present day distributed systems indicate the usefulness of a practical self-stabilizing system.

Self-stabilizing maxima finding has been considered by researchers for quite some time. Ghosh, and Lin [6] consider maxima finding in a ring network. Maxima finding in general networks have been addressed by many works such as [3, 2, 1]. Important problems to be dealt with in case of transient faults are, corrupted messages, transmission errors, and fake ids [1]. In [2], a special mechanism was developed to overcome the problem of fake ids, while [3] assumed the knowledge of an upper bound on the network diameter. We effectively deal with all the problems mentioned above, do not make any assumptions about the knowledge of a key parameter like diameter. It may be noted that maintaining the diameter of the network in a self-stabilizing way itself is a problem of interest. Afek, and Bremler [1] give a self-stabilizing leader election for general unidirectional networks. But they make the assumption of FIFO communication channels to prove a time bound of $O(n)$ for convergence where $n$ is number of nodes in the system. In our protocol, we do not make any assumption about the channel, and prove only the correctness of the protocol. Most leader election algorithms also construct a spanning tree rooted at the leader, and use this tree for communication, and co-ordination purposes. In such cases, the height of such a spanning tree constructed is important in worst case communication delay between leader, and a node. Our algorithm constructs a minimum depth spanning tree of the network rooted at the leader. Our algorithm is very simple, and easy to implement. In summary, in this paper, we present a self-stabilizing protocol for leader election in networks of any topology encompassing a broad transient fault model, and providing a very desirable property of constructing a minimum depth spanning spanning tree of the network in the process.

# 2 System Model

The system is modeled as a graph $G = (V, E)$ where $V$ is the set of nodes and $E$ is the set of edges between these nodes. The nodes correspond to processors in the system and an edge between two nodes represents a communication channel between corresponding processors. Each node $i$ has a unique id, $id$ which is assumed to be not corruptible. As assumed by Ghosh *et al* [6], it is assumed that each node knows another constant, $n$ which is the number of nodes in the network. These are reasonable assumptions made in many self-stabilizing algorithms.[1] Our algorithm elects the node with maximum id as the *maxima* or *leader*. To use our algorithm for dynamic situations like resource allocation, the id may be replaced by a function which evaluates the value of current resources held by the node.

We use the serial model of computation where a central daemon randomly selects one of the nodes with enabled guards. If the chosen node has multiple guards enabled, then one of the guards is randomly chosen. The assumption of a central daemon is mainly to simplify the proof. An algorithm to transform a self-stabilizing serial model algorithm to an equivalent self-stabilizing algorithm for asynchronous computing model has been discussed in the literature [7]. In the asynchronous computing model there is no central daemon and nodes execute the algorithm

[1]Ghosh *et al* [6] assume two non-corruptible integer data items namely $id$ and $n$, the number of processors in the ring.

asynchronously. Hence the assumption of a central daemon does not limit the applications of our algorithm.

## 3 The Algorithm

In this section, we present our algorithm. Our algorithm is represented as a set of guarded action statements. Each statement of the algorithm is written as $<$ *label* $>$: $C_i \longrightarrow A_i$ where $C_i$ is a guard or a predicate on the values of variables of a node $i$ and its neighbors, and $A_i$ is an action that should be performed by the node $i$ if this guard is satisfied. At each step in our algorithm, a central daemon evaluates guards of all nodes, selects randomly one node for which at least one guard is enabled. If more than one guard is enabled, then it also selects at random one of the enabled guards. The chosen node then executes the action corresponding to the enabled guard chosen by central daemon. In addition to its $id$ and $n$, each node maintains two integer variables $max$ and $dist$. These two variables are vulnerable to transient fault. In other words, in start state these variables can have any arbitrary values. $max$ of a node is its estimate of the maximum id in the system and $dist$ is the estimate of its distance from the node with maximum id. For each node $i$, $N_i$ represents the set of neighbors of the node $i$. As we do not make any assumptions of the nature of transient fault, a fault can take the system to any arbitrary state. This can be thought of as a starting point, and our algorithm stabilizes to a legitimate state. It is easy to note that such a fault model incorporates possibility of corrupted messages, transmission errors, and fake ids. The algorithm presented below.

**repeat**

$G0 : (i.dist = 0) \wedge (i.max \neq i.id)$
$\quad \longrightarrow i.max = i.id;$

$G1 : (i.dist \neq 0) \wedge (i.max = i.id)$
$\quad \longrightarrow i.dist = 0;$

$G2 : (i.dist > n - 1)$
$\quad \longrightarrow i.max = i.id;$
$\quad\quad i.dist = 0;$

$G3 : (i.max < i.id)$
$\quad \longrightarrow i.max = i.id;$
$\quad\quad i.dist = 0;$

$G4 : (i.dist > 0) \wedge (\nexists j \in N_i \,|\, (j.max = i.max) \wedge (i.dist = j.dist + 1))$
$\quad \longrightarrow i.max = i.id;$
$\quad\quad i.dist = 0;$

$G5 : (\exists\, j \in N_i \,|\, (j.max > i.max) \wedge (j.dist + 1 \leq n - 1))$
$\quad \longrightarrow i.max = j.max;$
$\quad\quad i.dist = j.dist + 1;$

$G6 : (\forall j \in N_i : j.max \leq i.max) \wedge (\exists k \in N_i \,|\, (k.max = i.max) \wedge (k.dist < i.dist - 1))$
$\quad \longrightarrow i.dist = k.dist + 1;$

**forever**

## 4 Description of the algorithm

The guards $G0$, $G1$, $G2$, $G3$ and $G4$ are termed as *correction steps.* They are mainly used to correct an erroneous state. The guards $G5$ and $G6$ are termed as *follow steps.* They are mainly aimed at maxima finding by setting a node's $max$ to the

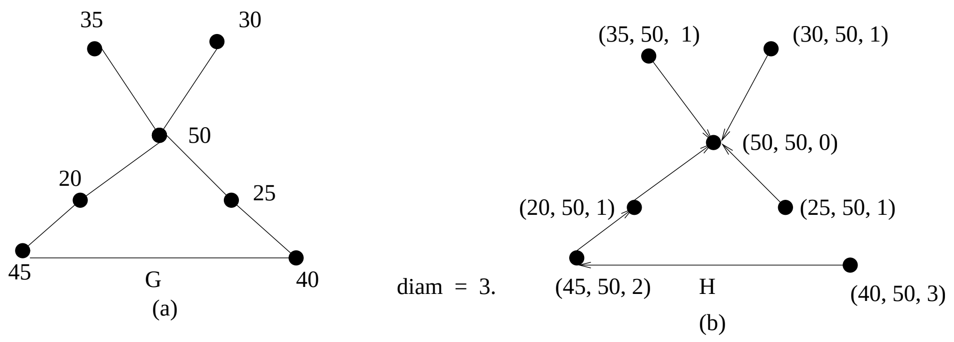


Figure 1: A follow graph with non-minimal depth.

value of *max* of a selected neighbor of the node and its *dist* to one more than that of the selected neighbor. Execution of a step is called a *move*.

The guards $G0$ and $G1$ are to remove inconsistencies in the local state of a node. They require the leader to have *max* equal to its *id* and *dist* to be zero. $G2$ requires the *dist* variable of a node to be less than or equal to the number of nodes minus one. $G3$ says that, the estimate of maximum id of a node should be at least as large as its own id. $G4$ says that, if a node is not a leader then, it should have a neighbor whom it has followed. Conceptually a node $i$ has followed a node $j$ if $((j \in N_i) \wedge (i.max = j.max) \wedge (i.dist = j.dist + 1))$. We term this as a *follow relation* between $i$ and $j$. In $G5$, a node $i$ follows a node $j \in N_i$ if $j$ has a higher estimate of maximum and $j$'s distance estimate is less than $n-2$. This ensures that $i$'s distance estimate is valid after executing the corresponding action. $G6$ requires that if a node can follow many nodes all of which have same estimate of maxima then it should follow the node with the least distance. We explain the need for $G6$ in the following paragraph.

Let us define a directed graph $H$ as follows: $H = (V', E')$ where $V' = V$ and $E' = \{(i, j) \in E \,|\, i\ has\ followed\ j\}$. We call $H$ as the *follow graph* of a system state. If there are many nodes that a node $i$ can follow, then one of them is randomly chosen to establish the corresponding follow edge $(i, j)$. It is easy to see that, when a leader is elected, the *follow graph* of the system state is a spanning tree of the network. We will now illustrate an undesirable scenario that may occur in the absence of $G6$. A leader may be elected, but the *follow graph* defined by the system state may not be of minimal depth.An example is shown in figure 1. Figure 1(a) shows a graph $G$ with diameter 3. The id of each node is shown beside it. Figure 1(b) shows the follow graph $H$ for the system in its current state. A triple $(id, max, dist)$ is shown beside each node. In this state the maxima is elected. However the depth of follow graph is 3, while minimum depth spanning tree rooted at node the with id 50 is 2. When $G6$ is present, the node with id 40 follows the node with id 25, instead of the node with id 45. Thus the depth of spanning *follow graph* is 2 which is minimal. So $G6$ is crucial for ensuring the property of minimal depth spanning tree when leader is elected.

We define a global predicate $GP$ which holds when a leader is elected and the algorithm has terminated. Let *maxnode* be the node with maximum id. Then

$$\begin{aligned}
GP1 &\equiv ((maxnode.max = maxnode.id) \wedge (maxnode.dist = 0)) \\
GP2 &\equiv (\forall j \neq maxnode, (j.max = maxnode.id) \wedge \\
&\qquad (\exists k \in N_j \,|\, j.dist = k.dist + 1)) \\
GP3 &\equiv (\forall j \neq maxnode, \nexists k \in N_j \,|\, (k.dist < j.dist - 1)) \\
GP &\equiv GP1 \wedge GP2 \wedge GP3
\end{aligned}$$

Our algorithm ensures that at termination, the follow graph of the system state is a minimum depth spanning tree rooted at *maxnode*. We prove this claim in the next section.

# 5 Safety and Liveness

The following lemmas are sufficient to prove safety and liveness properties of the system.

**Lemma 1.** *Let maxnode be the node with maximum id. When $GP$ is true the spanning tree defined by follow relations is the spanning tree of minimum depth rooted at maxnode.*

*Proof.* We prove the lemma by contradiction. Assume that $GP$ holds. Let the follow graph be a spanning tree $T1$ rooted at *maxnode*. Let $T2$ be the minimum depth spanning tree of the graph $G$ rooted at *maxnode*. Let $T2$ have depth *mindepth*. Let the depth of $T1$ be greater than *mindepth*. Consider a node $i_d$ in $T1$ at depth $mindepth + 1$. In $T2$, there must be a path of length $d \leq mindepth$ from *maxnode* to $i_d$. Let this path be $maxnode, i_1, i_2, \ldots, i_d$ where $i_k.dist = k$ for $k = 1 \ldots d$. Now let $i_j$ be the first node along the path $i_1 \ldots i_d$ which is at a depth greater than $j$ in $T1$. Now $i_{j-1} \in N_{i_j}$ and $i_{j-1}.dist = j - 1$. So $GP3$ does not hold for node $i_j$. This is a contradiction since $GP$ is true. □

**Lemma 2.** *(***Safety***) When $GP$ is true no guard is enabled for any node.*

*Proof.* Let *maxnode* be the node with maximum id. Clearly no guard is enabled for *maxnode*. Now consider any node $j \neq maxnode$. Clearly $G0$, $G1$, $G3$ are not enabled. $G4$ is not enabled because $GP2$ is true. By Lemma 1, $G2$ is not enabled. Since $i.max = maxnode.max\, \forall i$, $G5$ is also not enabled. $G6$ is not enabled because $GP3$ is true. Thus no guard is enabled for any node. □

**Lemma 3.** *(***Liveness***) If $GP$ is not true, then there exists at least one node for which at least one guard is enabled.*

*Proof.* We prove this lemma considering $GP1$, $GP2$ and $GP3$ separately.

1. $GP1$ does not hold.
   (a) $(maxnode.max = maxnode.id) \wedge (maxnode.dist \neq 0)$ : $G1$ is enabled for *maxnode*.
   (b) $maxnode.max < maxnode.id$ : $G3$ is enabled for *maxnode*.
   (c) $maxnode.max > maxnode.id$ :
      i. $maxnode.dist = 0$ : $G0$ is enabled for *maxnode*.
      ii. $maxnode.dist > 0$ :
      Consider the maximum length path $maxnode$, $i_1$, $i_2, \ldots, i_{last}$ such that each node in the path has followed its successor by the follow relation. Consider the node $i_{last}$.
         $i_{last}.dist > 0$ : Since the path is maximal length, there is no neighbor of $i_{last}$ whom it has followed. Hence $G4$ is enabled for the node $i_{last}$.
         $i_{last}.dist = 0$ : As *maxnode* has transitively followed $i_{last}$, $i_{last}.max = maxnode.max$. So from $maxnode.max > maxnode.id$ it follows that $i_{last}.max \neq i_{last}.id$. Thus $G0$ is enabled for the node $i_{last}$.

2. $GP1$ holds but $GP2$ does not hold.
Let set $S$ consist of $maxnode$ and all nodes $j$ with $j.dist \leq n - 1$ which have followed some node $k$ such that $k.max = maxnode.id$. That is, $S = \{maxnode\} \cup \{ j \mid (j.max = maxnode.max) \wedge (\neg G4 \text{ for } j) \wedge (j.dist \leq n-1) \}$. Let $S1 = V - S$. Since the graph is connected, there must exist nodes $j$ and $k$ such that $(j \in S \wedge j.dist < n-1)$ and $k \in S1$ and $j$, $k$ are neighbors.
   (a) $k.max < j.max$ : $G5$ is enabled for $k$.
   (b) $k.max > j.max$ : If $k.dist = 0$ then $G0$ is enabled for $k$. If $k.dist < n-1$ then $G5$ is enabled for $j$. Consider the case where $k.dist = n-1$. Let $k, i_1, i_2, \ldots, i_{last}$ be the maximal length path such that each node in the path has followed its successor. Following case 1(c)ii, either $G4$ is enabled for node $i_{last}$ or $G0$ is enabled for node $i_{last}$.
   (c) $k.max = maxnode.max$ : If $k.dist > n-1$ then $G2$ is enabled for $k$. Otherwise $k \notin S$ implies $G4$ is enabled for $k$.
3. $GP1$ and $GP2$ hold but $GP3$ does not hold.
If there exists a node $k$ such that $k.dist > n-1$ then $G2$ is enabled for $k$. If $i.dist \leq n-1 \forall i$ and $G6$ is not enabled for any node then $GP3$ is true and hence $GP$. But this is a contradiction since $GP3$ is false. □

$GP$ is a global predicate that is satisfied when a leader is elected. By Lemma 1 and Lemma 2 we have proved that irrespective of the start state a leader is elected when the algorithm terminates. We prove termination in the next section.

# 6 Termination

We complete the correctness proof by proving termination of the algorithm. For proving termination it suffices to show that every node in the system can execute only finite moves.

We define a *max-dist tuple* as a tuple $(max, dist)$. Two max-dist tuples are said to be *equal* iff both the components of the tuple are equal to each other. We define a relation *greater than* ($>$) between two tuples $T1 = (max_1, dist_1)$ and $T2 = (max_2, dist_2)$ as follows:

$T1 > T2$ iff
(i) $max_1 > max_2$ OR
(ii) $((max_1 = max_2) \wedge (dist_1 < dist_2))$.

**Lemma 4.** *If a node executes infinite moves then it executes infinite follow moves.*

*Proof.* Clearly a node can execute $G0, G1, G2, G3$ moves atmost once. Thus the only correction step that can be executed infinite times is $G4$. A $G4$ move by a node $i$ puts it in a state such that $(i.max = i.id) \wedge (i.dist = 0)$. Only a follow move can change $i.max$ or $i.dist$. Hence $i$ can execute the next $G4$ move only after it makes atleast one follow move. The same argument holds for subsequent $G4$ moves. Thus if a node executes $G4$ infinite times then it has to execute infinite follow moves. □

We define a set *repeated-follow-moves(i,j,a,b)* as a set of follow moves such that at each follow move in the set, $i$ follows $j$ when the max-dist tuple of $j$ is equal to $(a, b)$. If a set *repeated-follow-moves(i,j,a,b)* is infinite then we call it as *infinite-follow-moves(i,j,a,b)*.

**Lemma 5.** *For every node $i$ which makes infinite moves, there exists a node $j$ in its neighborhood such that $j$ makes infinite moves and* *(i) There exists a set $S$ of infinite-follow-moves$(i, j, max_j, dist_j)$,*

*(ii) there is a node $k \in N_j$ such that there exists a set $S1$ of infinite-follow-moves$(j, k, max_k, dist_k)$, and*

*(iii) $(max_k, dist_k) > (max_j, dist_j)$.*

*Proof.* Part(i) of the lemma is obvious as the number of possible values of $max$ and $dist$ is finite in the start state, and there must exist an *infinite-follow-moves*$(i, j, max_j, dist_j)$ for one or more nodes $j$. Let *infinite-follow-moves*$(i, j_1, max_{j_1}, dist_{j_1})$ be one such set. If there exists an *infinite-follow-moves*$(j_1, k, max_k, dist_k)$ for $k \in N_{j_1}$ then the lemma is true as it is imperative that $(max_k,\ dist_k) > (max_{j_1},\ dist_{j_1})$. If not, then for node $i$ to follow $j_1$ infinite times there must be another neighbor $j_2$ such that $i$ makes infinite oscillation of follow moves between nodes $j_1$ and $j_2$. However this will require that *infinite-follow-moves*$(i, j_2, max_{j_2}, dist_{j_2})$ exist. By $G5$ and $G6$ this requires $(max_{j_1},\ dist_{j_1}) > (max_{j_2},\ dist_{j_2})$, and $(max_{j_2},\ dist_{j_2}) > (max_{j_1},\ dist_{j_1})$ which is impossible. Hence one of $j_1$, $j_2$ must have a neighbor $k$ whom it follows infinite times. Let this node be $j_f$. Thus there must exist nodes $j_f$ and $k$ such that (i), (ii) and (iii) are true for $i$, $j_f$ and $k$. So the lemma is true. □

We observe that the relationship expressed in Lemma 5 is transitive.

**Theorem 1.** *No node can make infinite moves.*

*Proof.* Consider a node $i_1$ which makes infinite moves. Following lemma 5, there must be an infinite sequence of nodes $i_1$, $i_2$, $i_3, \ldots$ such that for all $x \geq 1$,
(i) *infinite-follow-moves*$(i_x, i_{x+1}, max_{i_{x+1}}, dist_{i_{x+1}})$ exists,
(ii) *infinite-follow-moves*$(i_{x+1}, i_{x+2}, max_{i_{x+2}}, dist_{i_{x+2}})$ exists, and
(iii) $(max_{i_{x+2}},\ dist_{i_{x+2}}) > (max_{i_{x+1}},\ dist_{i_{x+1}})$.
But the number of max-dist tuples in the system is finite. So this sequence cannot be an infinite sequence which is a contradiction. □.

**Theorem 2.** *(***Termination***) The algorithm computes leader in a self-stabilizing way.*

*Proof.* By $GP$, Lemma 2, Lemma 3 and Theorem 1 the algorithm elects the leader irrespective of starting state in finite time. □

# 7 Conclusion

In this paper, we have proposed a self-stabilizing algorithm for leader election of networks of any topology. We have proved algorithm's correctness, and termination properties. Our algorithm encompasses a very generic transient fault model, does not make assumptions about knowledge of properties like diameter of the netwrok, or channel properties like FIFO channels. Our algorithm constructs a spanning tree of minimum depth rooted at the maximal node. This is a very desirable property in applications where the spanning tree constructed is used for communication with the leader. Our simulation of the protocol suggest that our algorithm terminates in $O(n)$ time. We are currently investigating this property. We would also like to investigate if a self-stabilizing algorithm which terminates in $O(diameter)$ time is feasible without the knowledge of diameter.

# References


[1] Y. Afek and A. Bremler. Self-stabilizing unidirectional network algorithms by power supply. *Chicago Journal of Theoretical Computer Science*, December 1998.

[2] Y. Afek, S. Kutten, and M. Young. Memory-efficient self-stabilizing protocols for general networks. In *4th International workshop on distributed algorithms*, volume 484 of Lecture Notes in Computer Science, pages 15–28. Springer-Verlag, September 1990.

[3] A. Arora and M. Gouda. Distributed reset. *IEEE Transactions on computers*, 43:1026–1038, 1994.

[4] N Chen, H Yu, and S Huang. Self-stabilizing algorithm for constructing spanning trees. *Inf. Proc. Letters, Vol 39*, pages 147–151, 1991.

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

[6] X Lin and S Ghosh. Self-stabilizing maxima finding. *28th Annual Allerton Conference*, pages 662–671, 1991.

[7] M.Mizuno and M.Nesterenko. A transformation of self-stabilizing serial model programs for asynchronous parallel computing environments. *Information Processing Letters*, 66:285–290, 1998.

[8] M Schneider. Self-stabilization. *ACM computing surveys*, 25(1), March 1993.