Chicago Journal of Theoretical Computer Science

The MIT Press

Volume 1998, Article 3

7 December 1998

ISSN 1073–0486. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493 USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://mitpress.mit.edu/CJTCS/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *ftp://mitpress.mit.edu/pub/CJTCS*
- *ftp://cs.uchicago.edu/pub/publications/cjtcs*

The *Chicago Journal of Theoretical Computer Science* is abstracted or indexed in *Research Alert,®SciSearch,®Current Contents®/Engineering Computing & Technology*, and *CompuMath Citation Index.®*

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in computer science.

Self-Stabilizing Unidirectional Network Algorithms by Power Supply

Yehuda Afek Anat Bremler

7 December, 1998

Abstract

Power supply, a surprisingly simple and new general paradigm for the development of self-stabilizing algorithms in different models, is introduced. The paradigm is exemplified by developing simple and efficient self-stabilizing algorithms for leader election and either breadth-first search or depth-first search spanning-tree constructions, in strongly connected unidirectional and bidirectional dynamic networks (synchronous and asynchronous). The different algorithms stabilize in $O(n)$ time in both synchronous and asynchronous networks without assuming any knowledge of the network topology or size, where n is the total number of nodes. Following the leader election algorithms, we present a generic self-stabilizing spanning tree and/or leader election algorithm that produces a whole spectrum of new and efficient algorithms for these problems. Two variations that produce either a rooted depth-first search tree or a rooted breadth-first search tree are presented.

1 Introduction

1-1 A distributed system is self-stabilizing if the system automatically enters a global legal state in a bounded amount of time after the last fault or corruption has occurred—regardless of the local state of each of its processors, the level of RAM corruption in any processor, or the placement of messages on its communication channels. Once in a legal and correct state, the system remains in a legal state unless another fault or topological change has occurred. The notion of self-stabilization was introduced by Dijkstra in 1974 [Dij74] and since then many algorithms for different problems and configurations have been developed. Self-stabilizing algorithms for message passing or shared memory systems were developed for either unidirectional or bidirectional rings [Dij74, AB93, ALss, MOY96, MOOY92, Mul88, BP89, BGM90, Her90, IJ93, KP90, AEYH92, BGW89] and for bidirectional arbitrary topology networks [DIM94, AKY90, APSV91, AV91, BGW89, AG94b, AKM+93]. In this article, we develop simple and efficient self-stabilizing algorithms for unidirectional, arbitrary, topology-dynamic networks. The techniques developed here also produce new simple and efficient algorithms for the bidirectional case. In either case, our algorithms do not make any assumptions about the network size or the messages and variables used in the algorithms.

1-2 The major obstacle in designing unidirectional self-stabilizing algorithms is the lack of acknowledgments. Bidirectional communication is heavily used by the nodes in the bidirectional self-stabilizing system to compare the states of neighboring nodes and to check their consistency, as, for example, in the local checking algorithms [AKY90, APSV91, APSVD94].

1-3 In this paper, we overcome the lack of bidirectional communication with a new and surprisingly simple technique called *power supply*. Using this technique, we present leader-election algorithms for synchronous (Section 3) and asynchronous (Section 4) networks. Subsequently, we generalize these algorithms (Section 5) with a new observation of Collin and Dolev [CD94] to a family of algorithms either for electing a leader, or for constructing a spanning tree. Two versions of the general algorithm produce a breadth-first search (BFS) tree and a depth-first search (DFS) tree, with or without a predistinguished leader; other versions are possible. Although there could be as much as $O(E)$ corrupted messages in a global state of the system, the time complexities (stabilization times) for our algorithms are $O(n)$ without making any assumption about the size of the network. The space complexity (overhead) of our algorithms is $O(\log n)$ bits per node.

1.1 The Paper in a Nutshell, and Related Works

1.1-1 The design of self-stabilizing algorithms for unidirectional networks has started to receive attention only in recent years with the works of Mayer, Ostrovsky, and Yung [MOY96] and with Afek and Lev [ALss]. These works have designed algorithms mostly for unidirectional rings, which leaves the arbitrary topology open. Our work is motivated by these works and by the requirements posed by SDH/SONET unidirectional networks [ALss].

1.1-2 Only a few non-fault-tolerant distributed algorithms for unidirectional networks have been developed in the past; see, e.g., [GA84, AG94a, GKA83, GK84, Kut88, ELW90, Pet82, OW95, DKR82].

1.1-3 This paper proceeds from simple (Section 3) to more difficult (Sections 4 and 5). Let us enumerate the key ideas:

1. *The basics.* The basic algorithms developed are leader-election algorithms that elect the smallest id as a leader. However, in self-stabilization, simply choosing the smallest id is not safe, because a fake id that is smaller than all the ids in the network may falsely be chosen by all the nodes.

2. *Addressing fake ids.* A standard technique to overcome this problem is to assign each node with its distance from the node whose identity it has selected as a leader [Taj77, DIM94]. To ensure self-stabilization, every node periodically checks that its distance is one more than its parent distance (the parent is the neighbor through which the node has discovered the leader with the distance parameter it believes in, which is nil if the node under consideration is the root). Still, this principle by itself is not sufficient, because, for example, a false id might circulate around a cycle increasing its distance parameter without bound, and the problem could go undetected.

3. *Resolving a fake id's effects.* This problem has been overcome in several different ways: in [DIM94] a predefined tree subnetwork was assumed; in [AKY90] a special mechanism was developed to overcome the problem; in [APSV91, AV91] a reset protocol was invoked each time an inconsistent situation was detected; and in [AG94b], the knowledge of a bound b on the network diameter was assumed. In this paper, a new technique is suggested, which has the advantage over the above strategies in that it also works in unidirectional networks without making any assumptions ([AG94b] also works in unidirectional networks but requires the knowledge of a bound on the network size).

4. *Power supply.* "Power," the first basic idea in this paper, has two parts. First, a legal leader becomes a source of power which is disseminated, while fake identities may not produce power (a legal leader is a node that is the leader of itself). Second, to be captured by a new leader, a

node consumes a fixed amount of that leader's power. Hence, fake ids that have no source of power eventually disappear.

5. *The synchronous case.* The implementation of the "power" idea in a synchronous network is simple: to be captured by a new $\mathrm{id}_{\mathrm{min}}$, a node has to receive a message with $\mathrm{id}_{\mathrm{min}}$ in two consecutive rounds from the same neighbor and no smaller id from any other neighbor. However, if after being captured by id_x a node does not receive an id_x message in any one round, it immediately becomes the leader of itself.

6. *The asynchronous case.* The implementation of the above idea in an asynchronous network is problematic, because on the one hand, nodes in a self-stabilizing asynchronous network have to periodically transmit messages; and on the other hand, the transmission of such messages may "give" power to a fake id. This problem is solved here by distinguishing between two kinds of messages, *weak* and *strong*. Weak messages have no power, and are sent periodically between neighbors to ensure the consistency of the global state. Any inconsistency that is detected causes the detecting node to become the leader of itself. Strong messages, on the other hand, carry power. Only leader nodes periodically produce strong messages. Every other node relays a strong message to each of its neighbors only when it receives a strong message from its parent. To be captured by a new id, $\mathrm{id}_{\mathrm{min}}$, a node has to receive two strong messages with $\mathrm{id}_{\mathrm{min}}$ from the same neighbor, and at the same time, this id is smaller than any other id it receives.

7. *A generic self-stabilizing algorithm.* In this discussion we introduce another idea which is orthogonal to the power-supply concept. We replace the minimum distance parameter (point 2 above) by a general metric that may accommodate different parameters and rules for their update, e.g., the beautiful and simple parameter introduced by Collin and Dolev [CD94]. In combination with any of our other techniques (e.g., power supply), this new metric generates a DFS tree (instead of a BFS tree with the distance parameter) rooted at the elected leader. An example of a third metric is given in Section 5.

2 The Model

2-1 We consider a unidirectional strongly connected network with a set V of n nodes and a set E of unidirectional links [AG94a]. A unidirectional link is a point-to-point (node-to-node) communication line over which information may flow in only one direction. We make the standard and realistic assumption that each node v has a unique identity called id_v.

2-2 An incoming link of a node is a link directed into the node, and an outgoing link of a node is the link directed away from that node. In the asynchronous network, the number of messages that may be present on any link at the same time is bounded by a constant B (independent of the network size). This assumption is not only realistic, but is also necessary, as it is shown in [DIM91] that almost any nontrivial task cannot be solved in a self-stabilizing manner if link capacities are unbounded. However, when bounded capacity links are used, a deadlock may be formed unless messages are handled with care. In this article, we maintain a buffer for each outgoing link (incoming link) where the last two different messages that could not have been sent (received) because the link is too slow (too fast), are stored. When the link (processor) becomes available again, these messages are the first to be sent (processed). However, for the sake of clarity in describing the algorithms, we disregard this mechanism; i.e., we assume bounded capacity links exist, and that deadlocks are not formed. It is easy to see that the two models are equivalent (see the remark at the end of Section 4 for more details).

2-3 Our algorithms also recover from corrupted messages and transmission errors. During stabilization, after failures and topological changes stop, each link is assumed to reliably transmit messages from the tail node to the head node of the link. Each message that arrives at a node is tagged by the port number over which it arrives, and messages are processed in the order of arrival. Moreover, messages arrive at one end of a link in the order that they have been sent from the other end (FIFO); otherwise, bounded-time self-stabilization is prohibited.

2-4 Another assumption that we make (and without which no self-stabilizing message-passing algorithm may work in a dynamic network) is that each node knows which of its incoming links are up and operational and which are down. Otherwise, nodes might be stuck in the asynchronous case, waiting for messages over a link that is no longer operational [AAG87].

2-5 In Section 3, we describe our power-supply algorithm in a synchronous network. All the processors in a synchronous network receive, at the same time, an infinite sequence of evenly spaced clock ticks. In each clock tick (pulse), the processor, based on its local state and on messages received from its neighbors at the beginning of the pulse, makes a transition into a new state and may send a message to any of its neighbors. Each message sent immediately after a pulse is received by its destination before the next pulse. The time interval between two consecutive clock pulses is called a *round*.

2-6 In an asynchronous network, the processors operate at arbitrary rates which might vary over time, and the messages incur unbounded and unpredictable, but finite, delays.

2-7 The diameter of a network G whose set of nodes is V is defined as follows: $\text{diameter}(G) = \max_{u,v \in V} \text{dist}(u, v)$, where $\text{dist}(u, v)$ is the number of edges in the shortest directed path from u to v.

3 The Synchronous Unidirectional Power-Supply Algorithm

3-1 Here we present a simple algorithm for synchronous unidirectional networks (see Figure 1). It stabilizes in $O(n)$ rounds, and does not assume any bound on the diameter or on any other parameter of the network. (We remark that by a slight change in the algorithm, its stabilizing time may be reduced to be proportional to the length of the longest simple path in the network, which is $O(n)$ for general networks. However, this change leads to much more complex proofs, so we do not present it here).

3-2 In this algorithm, again the smallest node identity is elected as a leader. Fake ids are eliminated by using the distance parameter and the following two rules:

1. to remain under the leadership of a leader L at distance d, $d > 0$, the minimum leader message the node receives at each round should be L with distance $d - 1$; and

2. to be captured for the first time by a leader L at distance d, the minimum leader message the node receives in *two* rounds in a row should be L, with distance $d - 1$.

The first rule ensures that nodes owned by a fake leader and with the smallest distance parameter overall (which is necessarily larger than zero) at a round would abandon that leader in the next round. This ensures that the minimum distance parameter associated with a fake leader increases by one in each round. The second rule stipulates that the maximum distance parameter associated with a fake leader cannot grow by one in every round; instead, it can grow by one only every two rounds. It thus consumes the leader's power, because fake leaders do not have a power supply (a source for leader messages). Therefore, if in the initial faulty state the number of distinct distances associated with a fake leader is denoted as Δd, then within at most $2\Delta d$ rounds that fake leader id vanishes (all of its power is consumed).

3-3 Once all fake ids have been eliminated, the smallest id in the network captures all the nodes, each with the correct shortest distance to the elected leader.

```
Procedure for Node $v$
Type
    leader_info = record: [id, dist]
Var
    $\text{id}_v$                          {the unique id of node $v$, fixed by the hardware}
    leader, new, prev: of type leader_info
    m: message of type leader_info
    M: set of messages of type leader_info that have been received in the current round

Each round do
    M := M$\cup$\{[$\text{id}_v$ , 0]\};
    new.id := $\min_{m\in M}$ m.id;
    new.dist := $\min_{m\in M}$\{m.dist + 1|m.id = new.id\};
    If leader $\neq$ new Then
        If prev = new Then        {second time the node receives the new information}
            leader :=new;
        Else                       {first time the node receives the new information}
            leader :=[ $\text{id}_v$, 0 ]                    {Node$v$ becomes a self-leader};
    prev :=new                     {saving the information from the last round};
    send(leader record) on all outgoing links.
```

Figure 1: The synchronous algorithm

3.1 The Correctness of the Algorithm

3.1-1 For the proof of correctness, we consider the execution of the algorithm in the network following the last changes and faults; that is, we assume that the execution starts in an arbitrary state, and that no faults or topological changes occur during the execution.

3.1-2 Clearly, two rounds after the initial state, the variables (new, prev, and leader) at all the nodes hold values that were actually sent by their neighbors. In the first theorem, we prove that in $O(n)$ time after this state, all fake ids disappear. In the second theorem, we prove that $O(D)$ is the time after all the fake ids have disappeared, where D is the diameter of the network. The smallest id in the network is elected leader by all of the nodes.

3.1-3 In Theorem 1, we prove that fake leaders eventually disappear.

Theorem 1 *Eventually, all nodes in the variable* leader *have id* $\in$ ID*, where* ID $= \{id_v | v \in V\}$.

Let $f$id be a fake id in the network, i.e., $f\mathrm{id} \notin V$.

Definition 1 *The* heights group *of the fake id* fid *in a state of the system is:*

$$\mathrm{heights}(f\mathrm{id}) = \{\mathsf{leader}_v.\mathrm{dist} | \exists v \in G, \mathsf{leader}_v.\mathrm{id} = f\mathrm{id}\}$$

3.1-4 We claim that for any fake id $f$id, the size of heights$f$id is decreasing with time. In Lemma 1, it is proved that the size of heights$f$id may not increase, and in Lemma 2 it is proved that the size of heights$f$id decreases every two rounds.

3.1-5 Let X_v^i denote variable X at node v at round i.

Lemma 1 $|\mathrm{heights}^{r-1}(f\mathrm{id})| \geq |\mathrm{heights}^{r}(f\mathrm{id})|$

Proof of Lemma 1 The lemma follows from the code, since for each $d \in \mathrm{heights}^{r}(f\mathrm{id})$, $d \geq 2$, there must have been a $d-1 \in \mathrm{heights}^{r-1}(f\mathrm{id})$. Otherwise, no node would have distance d in the current round. Specifically, a node u whose leader is $f$id with distance d must have received in the beginning of round r the message: $[f\mathrm{id}, d-1]$ (by line 3). Hence, there must have been an incoming neighbor of u, v, such that in round $r-1$ $\mathsf{leader}_v := [f\mathrm{id}, d-1]$.

Proof of Lemma 1 □

Lemma 2 $||\mathrm{heights}^{r-2}(f\mathrm{id})| > |\mathrm{heights}^{r}(f\mathrm{id})|$.

Proof of Lemma 2-1 **Proof of Lemma 2** By Lemma 1, for every $d \in \text{heights}^r(f\text{id})$, there is a $d-2 \in \text{heights}^{r-2}(f\text{id})$. To prove the current lemma, we show that there is at least one value dm in $\text{heights}^{r-2}(f\text{id})$ for which there is no $dm+2$ in $\text{heights}^r(f\text{id})$. Let $dm = \max\{\text{heights}^{r-2}(f\text{id})\}$. Assume by contradiction that $dm+2 \in \text{heights}^r(f\text{id})$, and that v is a node with $\mathsf{leader}_v^r = [f\text{id}, dm+2]$.

Proof of Lemma 2-2 Clearly, in round $r-2$, $\mathsf{leader}_v \neq [f\text{id},\ dm+2]$.

Proof of Lemma 2-3 Hence there are two possible cases: either $\mathsf{leader}_v = [f\text{id}, dm+2]$ also in rounds r and $r-1$, or, only at round r. We show that in either case, $\mathsf{new}_v^{r-1} = [f\text{id}, dm+2]$, hence node v must have received a message $[f\text{id},\ dm+1]$ in round $r-1$. Thus there has been a u, an incoming neighbor of v, such that $\mathsf{leader}_u = [f\text{id},\ dm+1]$ in round $r-2$, a contradiction.

Proof of Lemma 2-4 In the first case, since $\mathsf{leader}_v^{r-1} = [f\text{id}, dm+2]$, node v performs line 5, and $\mathsf{leader}_v^{r-1} = \mathsf{new}^{r-1} = [f\text{id}, dm+2]$.

Proof of Lemma 2-5 In the second case, since $\mathsf{leader}_v^r = [f\text{id}, dm+2]$, node v performs line 5, and $\mathsf{leader}_v^r = \mathsf{new}^r = \mathsf{prev}^r = [f\text{id},\ dm+2]$. Since $\mathsf{prev}^r = \mathsf{new}^{r-1}$ (by line 9), $\mathsf{new}^{r-1} = [f\text{id}, dm+2]$.

Proof of Lemma 2 □

Proof of Theorem 1 From the two lemmas, it follows that the size of heights($f$id) decreases by at least one every two rounds. Hence, Theorem 1 holds.

Proof of Theorem 1 □

Corollary 1 *Within $O(n)$ rounds after the last fault or topological change, all fake ids disappear.*

Theorem 2 *At $O(D)$ rounds after all fake ids have been eliminated, the minimum id in the network is elected leader by all the nodes, where D is the diameter of the network.*

Proof of Theorem 2 Let ID_v be the smallest id in the network. The theorem follows by a simple induction on the rounds, because of the elimination of all fake ids in round r_0. Clearly, $\mathsf{leader}_v.id = \text{ID}_v$ in round r_0. In round r_0+2, every node u whose distance from v is one has v as its leader at distance one. In round r_0+2D, the leader of all the nodes is ID_v.

Proof of Theorem 2 □

Corollary 2 *The time complexity of this part is $O(D)$.*

3.1-6 Hence, the time complexity of the algorithm is $O(n)$. Also, $\Omega(n)$ is the lower bound on the time complexity of our algorithm, as is shown in the Appendix.

4 The Asynchronous Power-Supply Algorithm

4-1 A fundamental characteristic of asynchronous self-stabilizing algorithms is that nodes have to periodically exchange messages with their neighbors (using time-outs). Otherwise, the system could be placed in a global state in which each node is waiting for a message from another node. This fundamental characteristic breaks our power-supply algorithm, because every node in an asynchronous environment spontaneously generates an unbounded number of messages, regardless of the number of messages it receives.

4-2 Therefore, we introduce a new idea to implement the power-supply principle in an asynchronous network. We distinguish between two types of messages: *weak* and *strong*. Weak messages are periodically sent by each node to its neighbors, ensuring that neighboring nodes are in a consistent state and no node is stuck waiting indefinitely for a message from the other node. Strong messages, on the other hand, play the role of the power messages from the synchronous algorithm; that is, only leader nodes generate strong messages spontaneously, and each of the other nodes sends one strong message to each of its neighbors for every correct and consistent strong message received over its parent port-id (the port through which a leader has captured a node, by two consecutive strong messages; details below).

4-3 Specifically (the code is given in Figure 2), each node has a current_leader record with an id field and a distance field as in the synchronous algorithm, plus a parent pointer, which is either nil if the node is itself a leader or is pointing to one of its ports. A node that is owned by another id becomes a leader if its state is inconsistent with the neighbor's message, which happens in either of the following two cases:

1. it receives a message (weak or strong) through its parent port-id that is different from its current_leader; i.e., a message with an id different from the node's current_leader.id or with a distance different from leader.dist, or

```
Procedure at Node v:
Type
   leader_info = record: [id, dist]
Var
   $id_v$;                                  {the unique id of node $v$, fixed by the hardware}
   current_leader,prev,msg: of type leader_info;
   parent: port-id;
   set prev_ports of {port-ids};

 Upon receiving message (msg, mtype) arriving at incoming port-id $p$
1  if parent = nil then current_leader := [$id_v$, 0]                    {to be consistent};
2  if [$id_v$, 0] $\leq_{\text{lexic}}$ current_leader then current_leader := [$id_v$, 0]; parent := nil;
3  if ($p$ = parent) $\wedge$ (msg = current_leader)
4      then if (mtype = strong) then send_neighbors(strong);
5  if ($p$ = parent) $\wedge$ (current_leader $\neq$ msg)       {inconsistent message from the parent}
6      then current := [$id_v$, 0];
7           parent := nil;
8           send_neigbors(strong);
9  if (msg $<_{\text{lexic}}$ current_leader) then
10     if (mtype = strong) $\wedge$ (prev = msg) $\wedge$ ($p \in$ prev_ports)
                                         {the second lexicographically smallest message}
11          then current_leader := msg;                          {the node is captured}
12               parent = $p$;
13               send_neigbors(strong);
14          else current_leader := [$id_v$, 0]   {the first lexicographically smallest message};
15               parent := nil;
16               send_neigbors(strong);

17 if (msg $<_{\text{lexic}}$ prev) then case(mtype)              {updating prev and prev_ports}
18                        strong:prev = msg; prev_ports := {$p$};
19                        weak:  prev = [$id_v$, 0];prev_ports := $\emptyset$;
20 if (msg = prev) $\wedge$ (mtype = strong) $\wedge$ ($p \notin$ prev_ports) then prev_ports := prev_ports $\cup$ {$p$};
21 if (msg $>_{\text{lexic}}$ prev) $\wedge$ ($p \in$ prev_ports) then prev_ports := prev_ports \ {$p$};

22 for every $p \in$ prev_ports            {to make the algorithm work in a dynamic network}
23 if $p$ is not alive then prev_ports := prev_ports \ {$p$};
24 if (prev_ports = $\emptyset$) then prev := [$id_v$, 0];
25 if (parent is not alive) then current_leader := [$id_v$, 0]; parent := nil

26  Procedure send_neighbors(mtype)
27 send ( [current_leader.id, current_leader.dist+1], mtype) to all neighbors;

28  Upon time-out() at node $v$
29 if parent = nil then send_neighbors (strong);
30                 else  send_neighbors (weak);
```

Figure 2: The asynchronous power-supply algorithm

2. it receives a message (msg), weak or strong, through any port-id that is lexicographically smaller than current_leader; i.e., either msg.id is smaller than its current_leader.id, or msg.id equals the current_leader.id and msg.dist is smaller than leader.dist.

4-4 A node that has been captured by a certain leader will be captured by a new leader only if either the new leader identity is smaller, or the new leader identity is the same as the old one but the new leader comes with a smaller distance parameter (that is, the new leader's information is lexicographically smaller than the old leader's information).

4.1 The Principle of Power Supply

4.1-1 To be captured, *two* consecutive strong messages with the new lexicographically smaller information must be received through the same port-id (i.e., with only consistently weak messages received through the port-id in between them), and at the same time no lexicographically smaller message can arrive through any other port-id. The first lexicographically smaller message to arrive immediately changes the current_leader of the node to itself at distance zero, and only the second message changes the current_leader to the new information.

4.1-2 This principle ensures that strong fake-id messages eventually disappear from the network, since strong messages cannot flow in a cycle, and the number of strong fake-id messages is reduced for each node being captured by the fake id. On every path, the number of strong messages cannot increase, because a node sends a strong fake-id message only in response to receiving one. An important point for the proof of correctness is that whenever a node changes its current_leader, the node sends a strong message with its new current_leader. Thus all the neighbors of this node would notice that it went through a state change. In particular, whenever node v that is owned by *old* is being captured by a new leader, *new*, it assigns id_v to its current_leader in between these changes and sends strong messages containing id_v before sending the new strong messages.

4.1-3 The implementation of the above in the code (Figure 2) uses a prev variable to store the smallest message body received in recent message exchanges, and prev_ports, which is the set of port-ids through which this new information has arrived.

4.1-4 For the algorithm to operate correctly and in a self-stabilizing manner in a dynamic network, several local conditions have to be repeatedly checked; if they are found inconsistent, they should be corrected. These conditions are:

- The link connected to the parent port-id should be up. If the parent link is found to be down, then the node should become a leader of itself (line 25).
- If any port-id in prev_ports is found to be a port to a link that is down, it is removed from the set. If the set prev_ports becomes empty, then prev is reset (lines 22–25).
- If the parent of node v is nil, then current_leader has to be [id$_v$, 0] and vice versa (lines 1–2).
- If current_leader.id is larger than the node's id, then again current_leader is reset to [id$_v$, 0] (lines 1–2).
- Each message must conform to the expected syntax, and negative numbers are not allowed. A node that receives an illegal message becomes a leader of itself.

4.1-5 Since the asynchronous algorithm is an instance of the generic algorithm, its time complexity is $O(n)$, as we show for the generic algorithm. Similarly, the correctness of the algorithm follows from the proof of correctness for the generic algorithm given in Section 6.

4.2 A Remark about the Model

4.2-1 As stated in Section 2, the number of messages on a link in a certain state is bounded by B. Thus if either a tail node tries to transmit faster than the rate of the link, or if a receiving head node is too slow to receive the messages at the rate they arrive over the link, messages might be lost. For our algorithms, this does not pose a problem. We assume that at both end ports of a link there is a process that works as follows: at each end port, the link keeps a buffer with room for two messages. At the outgoing end (tail), whenever the algorithm produces messages at a rate higher than the link rate, the process keeps the last two different messages that were not sent. These messages will be sent out as if the two-message buffer were part of the link.

4.2-2 Similarly, at the receiving end, the process maintains a two-message buffer that contains only the last two different messages that have arrived and have not been processed by the algorithm. This ensures that if a node changes its state several times, the last change will never be lost, and each of the node's neighbors will notice that it went through a state change.

5 A Generic Power-Supply Algorithm

5-1 The two self-stabilizing algorithms presented in Sections 3 and 4, and most of the other algorithms known [DIM94, AKY90, APSV91, AG94b, AKM$^+$93], rely on the distance parameter, i.e., on the fact that each node selects the node closest to the leader and updates its distance to be one more. Yet, in [CD94], Collin and Dolev present a self-stabilizing algorithm that relies on another metric, which in turn produces a DFS tree rather than a BFS tree. These results suggest that perhaps there is a basic principle unifying these metrics. In this section, we develop a generic algorithm into which different metrics may be plugged, e.g., one of the above two, or new ones. An example of such a new metric is given below.

5-2 The general algorithm produces a whole spectrum of self-stabilizing algorithms for both unidirectional and bidirectional networks, and is given in Figure 4. The algorithm is a combination of the power-supply principle from the previous sections, with a general scheme to produce spanning trees. The BFS principle as in [Taj77] is one instance of the general scheme to construct a BFS spanning tree, while the Collin-Dolev principle given in [CD94] is another instance producing a DFS.

5-3 From any initial state, the generalization guarantees to stabilize in $O(n)$ time units if the underlying principle that ensures a tree structure does not send huge amounts of information (i.e., as long as the message size is kept to $O(\log n)$ bits, or in a model that allows sending large messages in one time unit). If messages are larger than the model can allow, then the time complexity might be larger.

5-4 Let us first describe the underlying principles and properties of the family of tree-producing schemes that fit our general algorithm. All of these schemes work according to the following general mechanism: each node that is a candidate for leadership has a unique value called the *zero* of that node. In the algorithms for constructing a tree rooted at a predistinguished node, only that predistinguished node is a candidate.

```
The Type Info
  case ALGORITHM:
    LE + BFS, LE + DFS, LE + FP, FP: Type info = record: [id, param];
    BFS, DFS:                        Type info = record: [param];

The value zero type info at node v:
  case ALGORITHM:
    LE + BFS: $zero := [\mathrm{id}_v, 0\ ]$;
    LE + DFS, LE + FP: $zero := [\mathrm{id}_v, \perp\ ]$;
    BFS:            if $v$ is the root then $zero := [0]$;
                                       else $zero := [\infty]$;
    DFS:            if $v$ is the root then $zero := [\perp]$;
                                       else $zero := [\infty]$;
    FP:             if $v$ is the root then $zero := [0, \perp]$;
                                       else $zero := [\infty, \perp]$;

If $v$ is a root then parent := nil;

Function next(selected of type info, $p$ of type port-id): info
  case ALGORITHM:
    FP, LE + FP, LE + DFS:return [selected.id, selected.param $\circ$ $p$ ];
    LE + BFS:             return [selected.id, selected.param + 1];
    BFS:                  return [selected.param+1];
    DFS:                  return [selected.param $\circ$ $p$];

Function select(selected of type info, msg of type info): info
  case ALGORITHM:
    BFS + LE, DFS + LE, BFS, DFS:if msg $<_{\mathrm{lexic}}$ selected
                                 then return msg;
                                 else return selected;
    FP, LE + FP: if $(\mathrm{id}_v \notin \text{msg.param}) \wedge$
                 $( (\mathrm{id}_v \in \text{selected.param}) \vee (\text{msg.id} < \text{selected.id}))$
                 then return msg;
                 else return selected;
```

Figure 3: The generic framework

5-5 The *zero* value of each candidate is fixed in hardware (i.e., in stable and reliable memory), and it is usually based on the node's unique identity. In the algorithm, each candidate tries to "convince" all other nodes to choose its *zero* value as their selected values, and thus to capture them. To do so, every candidate suggests that each of its neighbors should be its selected parent, by sending each neighbor a special value computed by applying a function called *next* on the *zero* value. Each node v selects (according to a particular selection rule) one of the suggestions it receives, assigns it to its selected variable, and selects the link over which it arrives as its parent. Node v transitively suggests its neighbors should join the same selected candidate by sending them a special message computed by again applying the function *next* on v's selected value. This process continues transitively until one candidate captures the entire network. The process thus described constructs a tree structure that traces the paths along which the *zero* value of the tree root has disseminated.

5-6 For such a scheme to generate a self-stabilizing algorithm when combined with the power-supply technique, it has to satisfy particular characteristics. Each such scheme has three components: (1) the *next* function, used to compute the suggestions; (2) the selection rule, which each node applies to choose its selected variable from the suggestions it receives; and (3) a set of *zero* values, which are all the *zero* values of nodes in the network. This set of *zero* values has to satisfy the following three properties:

1. No legal sequence of selected values along a path may cycle; that is, in any cycle of parent links and selected values, at least one node locally detects (by observing its predecessor selection and its own selection) that its selected value is wrong.

2. If there are no faults or erroneous values, then exactly one candidate node captures the whole network. This node does not select a parent link (its parent is nil).

3. If there are no faults or erroneous values, then the process reaches a fixed point: the network reaches a state after which no node changes its selection.

5-7 Any scheme that satisfies these properties reaches a stable state in which the parent links induce a rooted tree spanning the network. Different schemes' *next* functions, selection rules, and sets of *zero* values produce different trees. In this paper, three basic schemes are used that produce a BFS tree, a DFS tree, or an arbitrary tree.

5-8 A scheme that satisfies the above guidelines to construct a DFS tree was given by Collin and Dolev [CD94]. The *zero* of a root node in that variation is the symbol $\perp$, and the selected value of each node is a string of output port-ids along a simple path from the root (candidate for leadership) to that node. The selection rule selects the neighbor such that its *next selected* value (or sequence of link ports) is lexicographically smallest. Note that in this case the *next* function also takes the port-id leading to each neighbor as a parameter.

5-9 In another example of the generic algorithm, each node maintains the sequence of nodes on a path from the root to itself. In the generic implementation, each node v selects and extends the list of a neighbor whose list does not include v.

5-10 The different variations of the scheme are specified in Figure 3, for inclusion with the power-supply code in Figure 4. We present the parameterization of the *zero* values, the *next* function, and the selection rules to produce the following variations:

1. a leader-election algorithm that also produces a rooted breadth-first search tree (as denoted in the previous section as LE + BFS);

2. a leader-election algorithm that also produces a rooted *depth*-first search tree (denoted as LE + DFS);

3. an algorithm that produces a BFS tree, given a distinguished root (denoted as BFS);

4. an algorithm that produces a DFS tree, given a distinguished root (denoted as DFS);

5. a leader-election algorithm that produces an arbitrary rooted tree (denoted as LE + FP); and

6. an algorithm that produces an arbitrary tree, given a distinguished root (denoted as FP).

5-11 The various schemes satisfy each property in different ways. In the BFS and DFS schemes, the no-cycle property is satisfied, because the set of all possible suggestions and *zero* values is a total ordered set; and, for any value x, $\mathit{next}(x) > x$.

5-12 In the third scheme, FP, the no-cycle property is trivially satisfied since each suggestion is a string of ids, and the function *next* at node v simply appends id_v to v's selected string. A node does not select a suggestion that contains its own id.

5-13 The no-cycle property together with the power-supply technique ensures that any phony suggested value eventually disappears.

5-14 The formal proof of the algorithm's properties is given in Section 6.

5-15 Note that in the cases involving a predistinguished leader (BFS or DFS), it is unnecessary for each node in the system to have a unique id. The time complexity of the general algorithm is $O(n)$ (proof in Section 6). In the case of a BFS with a predistinguished leader, the time complexity is lower, $O(D)$, which is also the optimal time complexity for this problem.

6 Correctness of the Generic Asynchronous Algorithm

6.1 The Central Theorem

6.1-1 In this section we prove the main theorem.

Theorem 3 *Within $O(nB)$ units of time after the last topological change or erroneous state change, a stable tree spans the network.*

6.1-2 There are three major steps in the proof:

- First, we show that in linear time after the last change or error in the network, all erroneous and illegal strong messages disappear (Lemma 4), and are never generated again.
- Second, we show that in $O(n)$ time after the first step, erroneous or illegal weak messages do not exist, and are never generated again (Lemma 6).
- Third, we show that in $O(n)$ time after the second step, a stable rooted tree spans the network (Lemma 8).

6.1-3 Let us start with some definitions:

```
Procedure at Node $v$
Type
    info = record: [id, param]
Var
    selected,prev,msg: of type info;
    parent: port-id;
    set prev_ports of {port-ids};

 Upon receiving message (msg, mtype) arriving at incoming port-id $p$
1  if parent = nil then selected := $zero$                              {to be consistent};
2  if (select(selected, zero) = $zero$) then selected := $zero$; parent := nil
3  if ($p$ = parent) $\wedge$ (msg = selected)                                  {send a message}
4      then if (mtype = strong) then send_neighbors(strong);
5  if (($p$ = parent) $\wedge$ (selected $\neq$ msg))             {inconsistent message from the parent}
6      then selected := $zero$;
7             parent := nil;
8             send_neighbors(strong);
9  if (select(selected, msg) = msg) then
10     if (mtype = strong) $\wedge$ (prev = msg) $\wedge$ ($p \in$ prev_ports){the second selected message}
11            then selected := msg;                                    {the node is captured}
12                   parent = $p$;
13                   send_neighbors(strong);
14            else selected := $zero$;                                {the first selected message}
15                 parent := nil;
16                 send_neighbors(strong);

17 if (select(prev, msg) = msg) then case(mtype)       {updating prev and prev_ports }
18                                   strong:prev = msg; prev_ports := {$p$};
19                                   weak:  prev = $zero$;prev_ports := $\emptyset$;
20 if (msg = prev) $\wedge$ (mtype = strong) $\wedge$ ($p \notin$ prev_ports) then prev_ports := prev_ports $\cup$ {$p$};
21 if (select(prev, msg) $\neq$ msg) $\wedge$ ($p \in$ prev_ports) then prev_ports := prev_ports $\setminus$ {$p$};

22 for every $p \in$ prev_ports              {to make the algorithm work in a dynamic network}
23 if $p$ is not alive then prev_ports := prev_ports $\setminus$ {$p$};
24 if prev_ports = $\emptyset$ then prev := $zero$;
25 if (parent is not alive) then selected := $zero$;parent := nil;

26 procedure send_neighbors(mtype)
27 for every $p \in$ prev_ports
28                                        send (next(selected, $p$), mtype) to neighbor $p$;

29  upon time-out() at node $v$
30 if parent = nil then send_neighbors (strong);
31 else              send_neighbors (weak);
```

Figure 4: The generic power-supply algorithm

Definition 2 *Two functions are associated with each message m:* mtype(m), *the type of the message (strong or weak), and* msg(m), *the body of the message, which is of type* info *(see the code).*

Definition 3 *A message sequence, denoted by $\vec{m} = \{m_1 m_2, \ldots, m_k\}$, is a contiguous sequence of messages (strong and weak) on a link such that they all have the same* msg *value.*

Remark 1 *A message sequence can be empty.*

Definition 4 *We denote by* $\mathsf{in\_port}_v(w)$ *the port id through which the (wv) link enters v. We denote by* $\mathsf{out\_port}_v(w)$ *the port id through which the (vw) link leaves v.*

6.1-4 If we observe the edges that are covered by the union of message sequences that appear to have been originated from the same candidate, then this structure may look like a forest. However, in the proof, it is easier and much simpler to argue about the union of such sequences along a path, rather than along the edges of a tree. Below we formally define such sequences as *blocks*. Clearly, if we prove statements like "there are no more blocks with property p in the network," then it is true that the same holds for the corresponding trees with the same property. Therefore, it is enough to consider blocks, as defined below, rather then the more complicated trees.

Definition 5 *A block—denoted by* $\mathrm{bl} = \{\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$—*is a sequence of messages interleaved with zero or more nodes in one state of the system, such that the following hold (see Figure 5):*

1. *$\{v_0,\, v_1, \ldots v_l\}$ is a directed path.*
2. *For $i = 0, \ldots, l+1$, $\vec{m}_i = \{m_{i,1} m_{i,2}, \ldots, m_{i,k_0}\ \}$ is a message sequence on the link from v_i to v_{i+1}. Except for $\vec{m}_{l+1}$, each $\vec{m}_i$ contains all the messages on the corresponding link. If $\vec{m}_0$ is not empty, then m_{0,k_0} is the first message on the link entering v_0, and we denote by* begin_port *the port-id at node v_0 through which message m_{0,k_0} arrives. If $\vec{m}_0$ is empty, then* begin_port *is a port-id of some incoming neighbor of v_0. If $\vec{m}_{l+1}$ is not empty, then $m_{l+1,0}$ is the last message sent by v_l on the corresponding outgoing link and we denote by* end_port *the port-id at node v_l through which message $m_{l+1,0}$ was sent.*

3. *For* $i = 0, \ldots, l$, *for all* $m \in \vec{m}_i$, $\mathsf{msg}(m) = \mathsf{selected}_{v_i}$.
   For $i = l+1$, *for all* $m \in \vec{m}_{l+1}$, $\mathsf{msg}(m) = \textit{next}(\mathsf{selected}_{v_l}, \mathsf{end\_port})$.

4. *For* $i = 0, \ldots, l-1$, $next(\mathsf{selected}_{v_i}, \mathsf{out\_port}_{v_i}(v_{i+1})) = \mathsf{selected}_{v_{i+1}}$.

5. *For* $i = 1, \ldots, l$, $\mathsf{parent}_{v_i} = \mathsf{in\_port}_{v_i}(v_{i-1})$.
   For $i = 0, \mathsf{parent}_{v_0} = \mathsf{begin\_port}$.

6. *There is no* v *in* G *such that* $\{v\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$ *satisfies conditions 1–5, and there is no message* m *such that* $\{\{m\} \cup \vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$ *satisfies conditions 1–5.*

Observation 1 *The maximum number of nodes in a block is* n.

Observation 2 *Every block induces a simple directed path.*

6.1-5 Observation 2 follows from point 4 of Definition 5 and from the particular properties of the spanning-tree construction scheme. The no-cycle property has to be proved separately for each spanning-tree scheme that is used. It is easy to verify the property for the three major schemes we use: the BFS, DFS, and FP (see the discussion in the previous section).

Definition 6 *The potential(bl) of a block bl is the total number of strong messages in the block (see Figure 5).*

Corollary 3 *The potential of a block is at most* nB.

6.1-6 Were we proving the leader-election algorithm of Section 4, we would argue that blocks with fake ids eventually disappear. However, in the generic algorithm we have generalized the notion of id, and thus we define the notion of a *phony* block as follows.

Definition 7 *A block* $\{\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$ *is a phony block, if this block is a result of a "maliciously" erroneous initial state; that is, a block that was created after state* s_0 *and was truncated may not be considered to be a phony block. Formally, a block* $\{\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$ *is a phony block, if there is no way to assign values to* **selected** *fields of nodes* $v_{-h}, v_{-(h-1)}, \ldots, v_{-1}$ *such that* $\{v_{-h}, v_{-(h-1)}, \ldots, v_{-1}\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ *is a legal block and* $\mathsf{selected}_{v_{-h}}$ *equals* $zero_{v_{-h}}$ *(see Figure 5).*

Definition 8 *Let* $\{\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$ *be a block. Then* $m_{0,0}$ *is the tail of the block, or if* $\vec{m}_0$ *is empty, then* v_0 *is the tail of the block. Similarly,* $m_{l+1,k_{l+1}}$ *is the head of the block, or if* $\vec{m}_{l+1}$ *is empty, then* v_l *is the head of the block (see Figure 5).*

Definition 9 *A subblock is a subsegment of a block that satisfies all the conditions of Definition 5, except condition 6 (see Figure 5).*

Observation 3 *Every block is also a subblock.*

Preoof of Theorem 3-1 **Proof of Theorem 3 (The Formal Proof)** The proof argues about runs of the system that start in an arbitrary state s_0 and in which there are no failures or topological changes. Starting in state s_0, the system behavior is modeled by a run, which is an infinite sequence $q_0\pi_0 q_1\pi_1 \ldots$ of alternating states and atomic operations, such that $q_0 = s_0$. Each atomic operation π_i is either receiving and processing a message, and sending any resulting message, or a time-out event that results in sending messages. Each state includes a complete description of all the variables and messages in all the processors of the system. State q_{i+1} is the state of the system after applying operation π_i to state q_i.

Preoof of Theorem 3-2 For the purpose of time-complexity analysis, we assume that each message is delivered in at most one unit of time, i.e., one unit of time is the time it takes the slowest message to reach its destination.

Preoof of Theorem 3-3 We proceed by proving that following global state s_0, the system must progress through a sequence of global states that contains subsequence s_0, s_1, s_2, s_3, and s_4, such that in each of these states an additional global and *stable* property holds until in state s_4 the desired spanning-tree property stably holds. The sequence of stable properties that hold in the run suffixes starting at states s_0, s_1, s_2, s_3, s_4 are correspondingly as follows:

1. Starting with global state s_0, there are no failures or topological changes by assumption.

2. Starting with global state s_1, all messages and all state variables (parent, selected, prev,prev_ports) hold values that result from reception of messages sent after s_0. It is easy to see that state s_1 is at most two time units after state s_0.

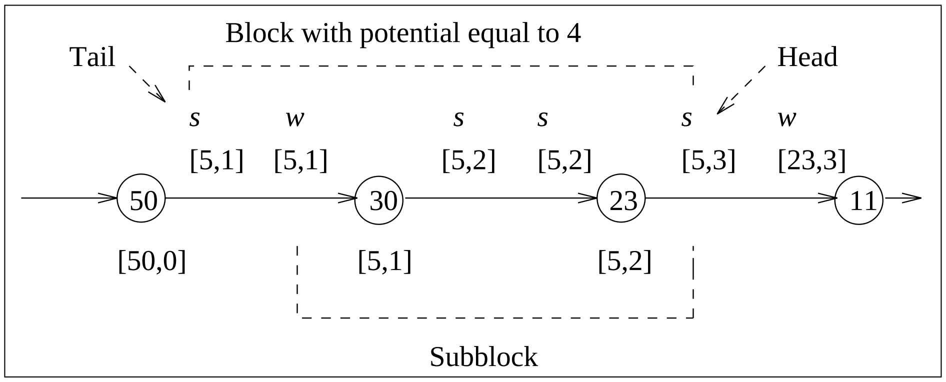

Figure 5: An example of a block, including the head and tail of a block, the potential, a phony block, and the subblock. The example is given in the framework of LE + BFS. Each node has an id. Under the node we see the node record [leader, distance], and similarly, the record of the messages on the links. We symbolize the type of message by s for strong and w for weak. The block in the figure is $\{m_{0,1} = [s, [5, 1]], m_{0,2} = [w, [5, 1]], v_1 = \{\mathrm{id}_v = 30, [5, 1]\}, m_{1,1} = [s, [5, 2]], m_{1,2} = [w, [5, 2]], v_2 = \{\mathrm{id}_v = 50, [5, 2]\}, m_{2,1} = [5, 3]\}$. The potential of the block is 4, which is the number of strong messages in the block. The message $m_{0,1} = [s, [5, 1]]$ is the head of the block, and the message $m_{2,1} = [5, 3]$ is its tail.
The sequence $\{m_{0,1} = [w, [5, 1]], v_1 = \{\mathrm{id}_v = 30, [5, 1]\}, m_{1,1} = [s, [5, 2]], m_{1,2} = [w, [5, 2]], v_2 = \{\mathrm{id}_v = 50, [5, 2]\}, m_{2,1} = [5, 3]\}$ is an example of a subblock, in the same way as any other suffix of the block.
The sequence $\{m_{0,1} = [w, [5, 1]], v_1 = \{\mathrm{id}_v = 30, [5, 1]\}, m_{1,1} = [s, [5, 2]], m_{1,2} = [w, [5, 2]], v_2 = \{\mathrm{id}_v = 50, [5, 2]\}, m_{2,1} = [5, 3]\}$ is an example of a subblock, in the same way as any other suffix of the block. (Sec 6.1)

3. Starting from global state s_2, there are no erroneous and illegal strong messages in the network.

4. Starting from global state s_3, there are no erroneous or illegal weak messages in the network.

5. Starting from global state s_4, a stable rooted tree spans the network.

Proof of Theorem 3 □

6.2 Proof Intuition

6.2-1 In the first (and major) step of the proof we show that phony blocks eventually disappear, by arguing that the lifetime of a phony block is bounded by the potential of the block (i.e., the potential of a phony block monotonically decreases). The potential of a phony block (by Definition 6) equals the maximum number of new nodes that the phony block may capture before disappearing (because the potential is the total number of strong messages, and the capture of a new node consumes at least one strong message). Once all the phony blocks disappear, the system reaches a stable legal state in $O(n)$ time.

6.2-2 Intuitively, the proof that phony blocks disappear after $O(nB)$ time is based on the following three points:

- P1: By Observation 2, a block cannot cycle.

- P2: A new node can be added to a block only if it is captured by two strong messages sent from the block (the power-supply principle). Therefore, in capturing a node, the phony block potential decreases by one.

- P3: The potential of a phony block cannot increase. It can only increase if one of the following conditions exists:

    1. it has the power supply to generate new strong messages (this is obviously not true in a phony block); or
    2. two or more phony blocks are united, thus creating a larger potential. But blocks cannot unite, because whenever a node changes its state, it first sends a message with its *zero* state to each of its neighbors. Upon receiving this message, each of the neighbors

that is at the same block as the node also goes through a change. Thus, an element (message or node state) separating blocks never disappears (until the blocks it separates disappear).

6.2-3 Based on points P1 and P2, in $O(nB)$ time after the initial state, the potential of a phony block reduces to zero. To observe this, let us track any strong message m on a phony block until it disappears. In each time-unit message, m propagates at least one more node along the phony block. The length of the phony block thus traversed is at most $n + nB + 1$, with n nodes in the original block, and $nB + 1$ that may be captured until its potential reaches zero. Therefore, after $O(nB)$ units of time, there is no strong message in any phony block, and the block potential is zero. Based on point P1, a phony block disappears in $O(n)$ time after its potential is equal to zero.

6.2-4 In Lemma 3, we prove that the potential of a phony block does not increase after state s_1 (point P3); i.e., distinct blocks cannot join together into a larger block. More formally, if in state q_{i+1} there is a block with m nodes, then in state q_i there is a corresponding block with at least $m - 1$ nodes and with at least as many strong messages. In the lemma we address all possible events and show that the potential does not increase in any of them.

6.2-5 Let X_v^i denote variable X at node v in state q_i.

Lemma 3 *Let q_{i+1} be a state, after state s_1. If in state q_{i+1} there exists a phony block $bl^{i+1} = \{\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_l v_l \vec{m}_{l+1}\}$ $l \geq 0$, then in state q_i there is a subphony block sbl^i, and exactly one of the following holds:*

1. *Node v_j receives a weak message: $sbl^i = \{\vec{m'}_0 v_0 \vec{m'}_1 v_1 \ldots \vec{m'}_{l-1} v_{l-1} \vec{m'}_l v_l \vec{m'}_{l+1}\}$. Only one message sequence, $\vec{m'}_j$, $0 \leq j \leq l+1$ in sbl^i, is not the same as in bl^{i+1}. Let $\vec{m'}_j = \{m_{j,1} \ldots m_{j,k_j-1} m_{j,k_j}\}$ in state q_i. Then $\vec{m}_j = \{m_{j,1} \ldots m_{j,k_j-1}\}$ in state q_{i+1}, where m_{j,k_j} is a weak message and $k_j \geq 1$.*

2. *Node v_j sends a weak message: $sbl^i = \{\vec{m'}_0 v_0 \vec{m'}_1 v_1 \ldots \vec{m'}_{l-1} v_{l-1} \vec{m'}_l v_l \vec{m'}_{l+1}\}$. Only one message sequence, $\vec{m'}_j$, $1 \leq j \leq l+1$ in sbl^i, is not the same as in bl^{i+1}. Let $\vec{m'}_j = \{m_{j,1} \ldots m_{j,k_j}\}$ in state q_i. Then $\vec{m}_j = \{m' m_{j,1} \ldots m_{j,k_j}\}$ in state q_{i+1}, where m' is a weak message.*

3. *Node v_j receives a strong message and sends a strong message: $sbl^i = \{\vec{m'}_0 v_0 \vec{m'}_1 v_1 \ldots \vec{m'}_{l-1} v_{l-1} \vec{m'}_l v_l \vec{m'}_{l+1}\}$. Only two message sequences, $\vec{m'}_j$*

and $\vec{m'}_{j+1}$, $0 \leq j \leq l$ in sbl^i are not the same as in bl^{i+1}. Let $\vec{m'}_j = \{m_{j,1} \dots m_{j,k_j-1} m_{j,k_j}\}$ and $\vec{m'}_{j+1} = \{m_{j+1,1} \dots m_{j+1,k_j}\}$ in state q_i. Then $\vec{m}_j = \{m_{j,1} \dots m_{j,k_j-1}\}$ and $\vec{m}_{j+1} = \{m' m_{j+1,1} \dots m_{j+1,k_j}\}$, where m_{j,k_j} and m' are strong messages.

4. *A node that is not in the block receives m_{l+1,k_l}, a strong message from the block: $sbl^i = \{\vec{m'}_0 v_0 \vec{m'}_1 v_1 \dots \vec{m'}_{l-1} v_{l-1} \vec{m'}_l v_l \vec{m'}_{l+1}\}$. Only $\vec{m'}_{l+1}$ in sbl^i is not the same as in bl^{i+1}. Let $\vec{m'}_{l+1} = \{m_{l+1,1} \dots m_{l+1,k_l-1} m_{l+1,k_l}\}$ in state q_i. Then $\vec{m}_{l+1} = \{m_{l+1,1} \dots m_{l+1,k_j-1}\}$ in state q_{i+1}, where m_{l+1,k_l} is a strong message.*

5. *A new node is added to the block after receiving a strong message from the block: $sbl^i = \{\vec{m'}_0 v_0 \vec{m'}_1 v_1 \dots \vec{m'}_{l-1} v_{l-1} \vec{m'}_l\}$. $\vec{m'}_l$ in sbl^i is not the same as in bl^{i+1}; and both v_l and $\vec{m}_{l+1}$ in bl^{i+1}, but not in sbl^i. Let $\vec{m'}_l = \{m_{l,1} \dots m_{l,k_l-1} m_{l,k_l}\}$ in state q_i. Then $\vec{m}_l = \{m_{l,1} \dots m_{l,k_j-1}\}$ in state q_{i+1}. In operation π_i, v_l changes its state and is captured to sbl^i after it receives the strong message m_{l,k_l} and sends the strong message m', where $\vec{m}_{l+1} = \{m'\}$.*

6. *The block did not change: $bl^{i+1} = sbl^i$.*

6.2-6 Before we prove the lemma, we will state and prove a few claims that are used in the proof of the lemma.

Claim 1 *If $\mathsf{parent}^i_w \neq \mathsf{nil}$ in state q_i, which is after s_1, then $\mathsf{selected}^i_w$ is equal to the body part of the last message received through port parent^i_w.*

Proof of Claim 1-1 **Proof of Claim 1** Let π_j be the last operation before state q_i in which a message m is received over port parent^i_w. By the time-out procedure and by the definition of state s_1, such a π_j exists. The value of parent_w does not change in all the states from q_{j+1} to q_i, because if it changes in π_k, then it changes either to nil or to the port over which a message was received in π_k (lines 11–13).

Proof of Claim 1-2 Since by the code, whenever the value of selected changes, the value of parent changes too, neither $\mathsf{selected}_w$ has changed in any state from state q_{j+1} to q_i. Thus $\mathsf{selected}^{j+1}_w = \mathsf{selected}^i_w$. To complete the proof, we need to show that $\mathsf{selected}^{j+1}_w = \mathsf{msg}(m)$. We consider two cases: either $\mathsf{selected}_w$ changes in π_j, or it does not change. In the former, $\mathsf{selected}^{j+1}_w = \mathsf{msg}(m)$ by lines 11 and 12; and in the latter, $\mathsf{selected}^{j+1}_w = \mathsf{msg}(m)$ by line 5.

Proof of Claim 1 □

Claim 2 *If* $\mathsf{prev}_v^i \neq zero_v$ *in state* q_i *which is after* s_1*, then* $\mathsf{prev\_ports} \neq \emptyset$*, and for every* p *such that* $p \in \mathsf{prev\_ports}_v^i$*,* prev_v^i *equals the body part of the last message received through port* p *at operation* π_j*, and* $p \in \mathsf{prev\_ports}_v^l$ *and* $\mathsf{prev}_v^i = \mathsf{prev}_v^l$ *for* $j+1 \leq l \leq i$*.*

Proof of Claim 2-1 **Proof of Claim 2** Since $\mathsf{prev}_v^i \neq zero_v$, then $\mathsf{prev\_ports}_v^i \neq \emptyset$ (by line 24).

Proof of Claim 2-2 Let π_j be the last operation before state q_i in which a message m is received over port p. Since port p is alive (lines 22 and 23) as argued in the previous proof, such a π_j exists.

Proof of Claim 2-3 Port $p \in \mathsf{prev\_ports}_v$ in all the states from q_{j+1} to q_i, because otherwise $\mathsf{prev\_ports}_v^i$ does not include p (lines 17–21).

Proof of Claim 2-4 Since (by the code) whenever the value of prev changes, the group of $\mathsf{prev\_ports}$ changes too (line 18), neither $\mathsf{prev\_v}$ has changed in any state from state q_{j+1} to q_i. Thus $\mathsf{prev}_v^i = \mathsf{prev}_v^{j+1}$. To complete the proof, we need to show that $\mathsf{prev}_v^{j+1} = \mathsf{msg}(m)$. We consider two cases: either prev_v changes in π_j, or it does not change. In the former, $\mathsf{prev}_v^{j+1} = \mathsf{msg}(m)$ by line 18; and in the latter, $\mathsf{selected}_w^{j+1} = \mathsf{msg}(m)$ by line 20.

Proof of Claim 2 □

Claim 3 *If in operation* π_i*, after state* s_1*, node* v *is captured by another node (i.e.,* v *performs lines 11 and 12 and* $\mathsf{selected}_v$ *is changed to a value different from* $zero_v$*), then:*

1. $\mathsf{selected}_v^i = zero_v$*, and*
2. *the* msg *part of the last message that* v *sends to each of its outgoing neighbors before operation* π_i *is equal to* $\mathsf{next}(zero_v, p)$*, where* p *is the port-id through which the message is sent.*

Proof of Claim 3-1 **Proof of Claim 3** Since node v is captured at operation π_i, then in state, q_i, by lines 9 and 10,

$$select(\mathsf{prev}_v^i, \mathsf{selected}_v^i) = \mathsf{prev}_v^i \tag{1}$$

If $\mathsf{prev}_v^i = zero_v$, then $\mathsf{selected}_v^i = zero_v$ (by Equation 1 and line 2), and we are done. Otherwise, $\mathsf{prev}_v^i \neq zero_v$, and hence $\mathsf{prev\_ports}_v^i \neq \emptyset$ (by line 24). From Claim 2 there exists an operation $\pi_k, k < i$ in which a message m is received over port p, $p \in \mathsf{prev\_ports}_v^i$. Let π_j be the last such operation before q_i. From Claim 2, $p \in \mathsf{prev\_ports}^l_v$, and $\mathsf{msg}(m) = \mathsf{prev}^i_v = \mathsf{prev}^l_v$ for $j+1 \leq l \leq i$.

Proof of Claim 3-2 To complete the proof of the first part of the claim, we prove that if the value of $\mathsf{selected}_v$ has changed in π_l, $j+1 \leq l \leq i$, then in the last such change $\mathsf{selected}_v$ was set to $zero_v$; otherwise it was set to $zero_v$ in π_j, and thus $\mathsf{selected}^i_v = \mathsf{selected}^j_v = zero_v$. To prove this, consider an operation π_r, $j \leq r < i$ such that π_r is the last time before π_i that the value of $\mathsf{selected}_v$ is changed. Notice that such an operation exists.

Proof of Claim 3-3 Assume, to the contrary, that there is no such operation π_r. Then, in π_j, v receives a message m such that $\mathsf{msg}(m) = \mathsf{prev}^i_v$ and select(prev^i_v, $\mathsf{selected}^i_v$) = select(prev^i_v, $\mathsf{selected}^j_v$) = prev^i_v. Hence $\mathsf{selected}_v$ must have changed its value in π_j by lines 9–16, in contradiction to the fact that there is no change in the value of $\mathsf{selected}_v$ in any operation $j \leq r < i$.

Proof of Claim 3-4 By definition of π_r, $\mathsf{selected}^{r+1}_v = \mathsf{selected}^i_v$. The value of $\mathsf{selected}_v$ changes in π_r to either prev^r_v or $zero_v$ (either in line 11 or lines 6–14).

Proof of Claim 3-5 In the latter case, $\mathsf{selected}^{r+1}_v = \mathsf{selected}^i_v = zero_v$, and the claim follows. We claim the former is impossible, i.e., $\mathsf{selected}_v$ may not change to prev_v in π_r. Otherwise, $\mathsf{prev}^{r+1}_v = \mathsf{selected}^{r+1}_v$, and since $\mathsf{prev}^{r+1}_v = \mathsf{prev}^i_v$, then $\mathsf{prev}^i_v = \mathsf{selected}^i_v$, in contradiction to Equation 1.

Proof of Claim 3-6 In π_r, node v also sends a message with the msg part equal to $\mathsf{next}(zero_v, p)$ (lines 8 or 16). In all the operations $r < k < i$, $\mathsf{selected}^k_v = zero_v$, and v can send a message only by performing a procedure time-out (lines 28–30), such that the msg part equals $\mathsf{next}(\mathsf{selected}^k_v, p) = \mathsf{next}(zero_v, p)$; hence the second part of the claim holds.

Proof of Claim 3 □

Claim 4 *Let π_l be an operation after s_1 but before state q_i. If in π_l v sends to its outgoing neighbor w message m_l, such that $\mathsf{msg}(m_l) = \mathsf{next}(zero_v, \mathsf{out\_port}_v(w))$, and if either*

1. *message m_l is the last message that arrives at w before q_i, or*

2. *message m_l has not yet arrived at w in state q_i,*

then in state q_i there is no phony block bl^i such that both v and w are $\in bl^i$.

Proof of Claim 4-1 **Proof of Claim 4**
Case 1: Let π_j be the operation in which message m_l is received by node w.

Proof of Claim 4-2 If in q_i there is a block bl^i such that $v, w \in \mathrm{bl}^i$, then $\mathsf{parent}^i_w = \mathsf{in\_port}_w(v)$, and from Claim 1, $\mathsf{selected}_{w^i}$ equals the msg part of m_l, the last message that was received from the parent port. Hence $\mathsf{selected}^i_v = \mathsf{next}(zero_v, \mathsf{out\_port}_v(w))$, a contradiction to the fact that bl^i is a phony block.

Proof of Claim 4-3 Case 2: Message m^l has not yet been received by w. Assume the opposite, that $v, w \in \mathrm{bl}^i$. Thus $\{v\vec{m}w\} \subseteq \mathrm{bl}^i$, $m^l \in \vec{m}$. By the definition of a block, $\mathsf{selected}^i_w = \mathsf{msg}(m^l) = \mathsf{next}(zero_v, \mathsf{out\_port}_v(w))$, and again this is a contradiction to the fact that the block is a phony block.

Proof of Claim 4 □

Proof of Lemma 3-1 **Proof of Lemma 3** Let us prove that subblock sbl^i must exist in state q_i. To prove this, assume such an sbl^i does not exist.

Proof of Lemma 3-2 Then if the $\mathrm{tail}(\mathrm{bl}^{i+1})$ element exists in q_i, then let e be the first element (message or node) along the path induced by bl^{i+1} that may not be a member of sbl^i because either it has a different selected (when e is a node) or an inconsistent parameter (when e is a message). There are four cases to consider:

1. $\mathrm{tail}(\mathrm{bl}^{i+1})$ does not exist in q_i,
2. e is a node,
3. e is a message, and
4. e does not exist.

A general remark: by the code, an operation π_i is exactly one of the following:

1. sending one new message (by procedure time-out),
2. receiving one message,
3. receiving one message and sending one new message without changing the node state, and
4. receiving one message, changing the state of node v, and sending one new message from node v.

Proof of Lemma 3-3 Case 1: Tail(bl^{i+1}) does not exist in q_i. If tail(bl^{i+1}) is a node, then this node has changed its selected in π_i. If tail(bl^{i+1}) is a message, it must have been generated in π_i. Let us consider each of the two subcases (tail(bl^{i+1}) is a node, or a message).

Proof of Lemma 3-4 In the former subcase, let v be the node captured in π_i. By Claim 3, if w is an outgoing neighbor of v, then the last message that v sent before state q_i was message m, such that $\mathrm{msg}(m) = \mathsf{next}(zero_v, \mathsf{out\_port}_v(w))$.

Proof of Lemma 3-5 By Claim 4, v and w may not be in the same phony block in state q_{i+1}. Since v is the tail of block bl^{i+1}, bl^{i+1} contains only one node, v. In operation π_i, node v was captured after receiving a message m', and sent a strong message, $\widehat{m}$, with $\mathsf{msg}(\widehat{m}) = \mathsf{next}(\mathsf{msg}(m'), \mathsf{out\_port}_v(u))$, where u is an outgoing neighbor of v (by lines 11–13). Thus we are at case 5 of Lemma 3.

Proof of Lemma 3-6 In the latter case, in operation π_i, some node v generates message m, which is the tail of bl^{i+1}. We claim that this case is impossible. Message m is sent by v to outgoing neighbor w using the procedure $\mathsf{send\_neighbors}$. By performing this procedure, node v sent the message $\mathsf{next}(\mathsf{selected}_v^{i+1}, \mathsf{out\_port}_v(w))$. Hence from the definition of a block and the fact that $m \in \mathrm{bl}^{i+1}$, $v \cup \mathrm{bl}^{i+1}$ is also a block, a contradiction to the fact that bl^{i+1} is a block (violating condition 6 of the block definition (Definition 5)).

Proof of Lemma 3-7 Case 2: e is a node. Thus $\mathsf{selected}_e^i \neq \mathsf{selected}_e^{i+1}$, and node e was captured in operation π_i (lines 11 and 12). By Claim 3, $\mathsf{selected}_e^i = zero_e$, and before state q_i, the last message e sent to its outgoing neighbor w was a message such that $\mathsf{msg}(e) = \mathsf{next}(zero_e, \mathsf{out\_port}_e(w))$. From Claim 4, e and w cannot be at the same phony block at state q_{i+1}; therefore we are again at case 5 of the lemma.

Proof of Lemma 3-8 Case 3: e is a message. If e is a weak message, then the message was generated in π_i. A weak message can be generated only by time-out (lines 29–31). Thus we are at case 2 of the lemma.

Proof of Lemma 3-9 If e is a strong message, then the message was created in π_i by some node v. We claim that $v \in \mathrm{sbl}^i$ and $v \in \mathrm{bl}^{i+1}$. If $v \in \mathrm{bl}^{i+1}$ and $v \notin \mathrm{sbl}^i$, then e is that node v and not the message, a contradiction. If $v \notin \mathrm{bl}^{i+1}$, then in state q_{i+1} the message $\mathsf{msg}(e) = \mathsf{next}(\mathsf{selected}_v^{i+1}, l)$ where l is the port id of the link on which e is sent (procedure send-neighbors). By the definition of block, $v \cup \mathrm{bl}^{i+1}$ is a block, in contradiction to condition 6 of Definition 5. Therefore, $v \in \mathrm{sbl}^i$ and $v \in \mathrm{bl}^{i+1}$. Thus $\mathsf{selected}_v^i = \mathsf{selected}_v^{i+1}$. A strong message may be sent without changing the status of a node by performing:

1. line 29 (time-out), but then $\mathrm{parent}_v = \mathrm{nil}$ and $\mathsf{selected}_v^{i+1} = zero_v$ (from

line 1), and this is a contradiction to the fact that the block is a phony block; and

2. lines 3 and 4, which corresponds to case 3 of the lemma.

Proof of Lemma 3-10 Case 4: e does not exist. In this case, we should check if there is an element induced by sbl^i that is not a member of bl^{i+1}. Let d be the first element (message or node) along the path induced by sbl^i which is not a member of bl^{i+1}. We consider two cases: (1) d is a node; and (2) d is a message. If d does not exist, then we are in case 6 of the lemma, $\mathrm{sbl}^i = \mathrm{bl}^{i+1}$.

Proof of Lemma 3-11 *Subcase 1: d is a message.* In operation π_i, some node v received d. If $v \in \mathrm{sbl}^i$, then if d is a strong message we are at case 3 of the lemma (lines 3 and 4), and if d is a weak message, then we are at case 1 of the lemma (a weak message can only change the selected of a node to zero). If $v \notin \mathrm{sbl}^i$, then d is the head of sbl^i. If $v \in \mathrm{bl}^{i+1}$, we are at the case that a new node is captured, which is case 5 of the lemma. If $v \notin \mathrm{bl}^{i+1}$, then only this message disappeared; thus we are at case 1 of the lemma if it is a weak message, or case 4 if it is a strong message.

Proof of Lemma 3-12 *Subcase 2: d is a node.* We show that in this case, $d \in \mathrm{sbl}^i$, but $d \notin \mathrm{bl}^{i+1}$, which is impossible; thus, this entire case is impossible. If d is the tail of sbl^i, then without loss of generality, sbl^i is defined as the subblock of $\mathrm{sbl}^i \setminus \{v\}$. If the head of sbl^i is node x but $x \notin \mathrm{bl}^{i+1}$, then without loss of generality, sbl^i is defined as the subblock of $\mathrm{sbl}^i \setminus x$. Hence $\mathrm{head}(\mathrm{sbl}^i) \in \mathrm{bl}^{i+1}$, and $\mathrm{tail}(\mathrm{sbl}^i) \in \mathrm{bl}^{i+1}$. Therefore without loss of generality, d is not the head or the tail of sbl^i and there exists a node or a message $r \in \mathrm{sbl}^i$ which is the following node or message along the block sbl^i, and also $r \in \mathrm{bl}^{i+1}$. Similarly, there exists a node or a message $y \in \mathrm{sbl}^i$ that is a previous node or message along the block sbl^i, and also $y \in \mathrm{bl}^{i+1}$. In operation π_i, d changed its $\mathsf{selected}_d$ or parent_d so that $d \in \mathrm{sbl}^i$ but $d \notin \mathrm{bl}^{i+1}$, however, the state of r and y did not change; thus it is easy to see that according to the definition of a block it cannot be that both r and y are in the same block bl^{i+1}.

Proof of Lemma 3 □

Corollary 4 *Let π_i be an operation after state s_1. Then potential(sbl^i) $\geq$ potential(bl^{i+1}).*

6.2-7 In Lemma 4, we prove that after $O(nB)$ time the potential of any phony block reduces to zero. More specifically, we show that all the strong messages in a phony block disappear after (at most) that much time. To do that, we pick an arbitrary message on a phony block, track its traversal of the block, and argue that such a traversal cannot last more than $O(nB)$ units of time. For the purpose of the proof, a strong message that is relayed by a node from its in-port to its out-port is considered the same message.

6.2-7 In the formal proof, we follow the distance between the tracked message and the head of the block (defined as the *remainder* in Definition 10). However, this process and definition is problematic: in continuing from the location of a specific message, a block may branch into several blocks, because the overall structure is that of a tree. Thus, the specific block traversed by the message is not well defined, and neither is the remainder of the message. To overcome this difficulty, we break the proof into three steps. In the first two steps of the proof, we run the algorithm forward and then observe the block in a backward execution of the same run. This solves our problem, because when the algorithm is run backwards, every block has a unique source subblock (based on Lemma 3). Then in the third step, we make the arguments necessary for the proof on the sequence of steps and blocks defined in the second step.

6.2-8 The three steps of the proof are thus:

1. Run the algorithm forward $nB + n + 3$ time units, and assume to the contrary that there exists a block with a nonzero potential, and hence there is a strong message in it.

2. By a backward simulation of the run on this block and this message in previous states, build the progression of the block and the traversal of that message in different states.

3. Prove on the forward sequence of blocks that the remainder of the message in that block, $nB + n + 3$ time units after the initial step, is zero. This last step is based on three points:

   (a) The remainder of the message in the initial state is at most $n + 1$.

   (b) At least $nB + n + 3$ times, the remainder of the message decreases by one, since at each time unit the message advances at least one more hop in the block.

   (c) The number of times the remainder of the message increases by one is at most $nB + 1$, the initial potential of the block. The correctness of the third statement is based on the following points:

- the remainder of the message increases if a new node is captured by the block;
- the block loses one strong message for every node that is captured by the block (except perhaps for the first node that is captured to the block; see Claim 5);
- the number of strong messages in the block is at most $nB+1$; and
- by Lemma 3, the number of strong messages in a block cannot increase.

Lemma 4 *In $O(nB)$ time units after state s_1, the potential of every phony block is zero.*

6.2-9 Before we prove the lemma, let us discuss some definitions.

Definition 10 *Let $bl = \{\vec{m}_0 v_0 \vec{m}_1 v_1 \ldots \vec{m}_{l-1} v_{l-1} \vec{m}_l v_l \vec{m}_{l+1}\}$ be a subblock. Define for each message $m \in \vec{m}_j$, $remainder(m, bl) = l + 1 - j$.*

Observation 4 *In any state, for every message m and subblock bl, $0 \leq remainder(m, bl) \leq n + 1$.*

Let π_i be an operation after s_1. Let bl^{i+1} be a phony block in state q_{i+1}.

Definition 11 *The subsource of block bl^{i+1} is the subblock sbl^i in state q_i, as defined in Lemma 3 for block bl^{i+1}.*

Definition 12 *The origin of message m^{i+1} is the strong message m^i in sbl^i where:*

1. *if in π_i, v sends m^{i+1}, then m^i is the message that in response to its reception v sent m^{i+1}; and*
2. *otherwise, $m^i = m^{i+1}$.*

Definition 13 *The source block of block bl^{i+1} is block bl^i, whose suffix is the subsource of block sbl^i.*

Observation 5 *$Remainder(m^i, sbl^i) = remainder(m^i, bl^i)$.*

The observation follows from the fact that the subsource block sbl^i is a suffix of the source block bl^i.

Observation 6 *$Potential(bl^{i+1}) \leq potential(bl^i)$.*

The observation derives from the following facts:

- $potential(sbl^i) \leq potential(bl^i)$ since sbl^i is the suffix of bl^i; and
- $potential(bl^{i+1}) \leq potential(sbl^i)$ (by Corollary 4).

Corollary 5 *Let m^i be the origin of m^{i+1}. Then there are three possible values for $remainder(m^{i+1}, bl^{i+1})$ as a function of $remainder(m^i, bl^i) - 1$:*

1. *$remainder(m^{i+1}, bl^{i+1}) = remainder(m^i, bl^i) - 1$, if in π_i message m^i was relayed by a node and sent on the next link on the block;*
2. *$remainder(m^{i+1}, bl^{i+1}) = remainder(m^i, bl^i) + 1$, if in π_i block bl^i, in which message m^i resides, captures a node;*
3. *$remainder(m^{i+1}, bl^{i+1}) = remainder(m^i, bl^i)$, which happens in either of the following two cases:*
    - *The head of bl^i is message m^i, and in π_i message m^i captures a new node and m^{i+1} is relayed by the captured node. In this case, the remainder of the head message remains zero, and hence $remainder(m^{i+1}, bl^{i+1}) = remainder(m^i, bl^i) = 0$. This case is considered to be the simultaneous occurrence of the first two possibilities.*
    - *$remainder(m^{i+1}, bl^{i+1}) = remainder(m^i, bl^i)$ if neither of the first two cases occurs. Then in π_i no new node is captured by bl^i, and π_i does not affect m^i.*

Corollary 5 is based on Lemma 3 and Observation 5.

Definition 14 *The progression of bl^w from state q_k to state q_w, $k \leq w$, is the series of blocks $bl^k bl^{k+1} \dots bl^w$ at states $q_k q_{k+1} \dots q_w$, such that for $k \leq i \leq w$, bl^i is the source block of bl^{i+1}.*

Remark 2 *Note that the progression of a block is defined backwards; i.e., bl^w is the first step, and the source relationship between bl^i and bl^{i+1} continues the definition from $i+1$ to i.*

Definition 15 *The strong message m^w traversal of block bl^w from state q_k to state q_w, $k \leq w$, is the series of strong messages $m^k, m^{k+1} \dots m^w$ at states $q_k, q_{k+1} \dots q_w$, such that for $k \leq i \leq w$, m^i is the origin of m^{i+1}, $m^i \in bl^i$, and $m^{i+1} \in bl^{i+1}$, and bl^i is the source block of bl^{i+1}.*

Remark 3 *Note that as in the definition of block progression, the message-traversal definition is also inductively backwards; i.e., the basis of the induction is message m^w and block bl^w, and the step of the induction is the origin relationship between m^i and m^{i+1}.*

6.2-10 Before we start the formal proof of Lemma 4, we will state and prove a basic claim that corresponds to the second bulleted point (at the end of the intuition paragraph before Lemma 4). Let bl^w be a block in state q^w, and let $\mathrm{bl}^0, \mathrm{bl}^1, \dots, \mathrm{bl}^w$ be the progression of the block from state q^0 to q^w. Let $\{\pi_{i_1}, \pi_{i_2}, \dots, \pi_{i_k}\}$ be a maximal series of operations such that for every j, a new node is captured by bl^{i_j-1} in π_{i_j}.

Claim 5 *For every j, $0 \leq j \leq k-1$, there exists an operation π_l, $i_j < l < i_{j+1}$ such that $potential(bl^{l+1}) \leq potential(bl^l) - 1$.*

Proof of Claim 5-1 **Proof of Claim 5** Let v be the node that is captured by subsource block bl^{i_j} in operation π_{i_j}. Let u be the node captured by source block $\mathrm{bl}^{i_{j+1}}$ in operation $\pi_{i_{j+1}}$. Node u is the outgoing neighbor of v. This follows from the fact that the series of $\{\pi_{i_1}\pi_{i_2} \dots \pi_{i_k}\}$ is maximal, and from the definition of the progression $\mathrm{bl}^0, \mathrm{bl}^1 \dots \mathrm{bl}^w$.

Proof of Claim 5-2 The last message that v sends to u before operation π_{i_j} is $m1$, such that $\mathsf{msg}(m1) = \mathsf{next}(zero_v, \mathsf{out\_port}_v(u))$ (by Claim 3). In operation π_{i_j}, node v sends to u a strong message, $m2$, such that $\mathsf{msg}(m2) = \mathsf{next}(\mathsf{selected}_v, \mathsf{out\_port}_v(u))$ where $m2 \in \mathrm{bl}^{i_j+1}$ (case 5 of Lemma 3). Hence $\mathsf{msg}(m2) \neq \mathsf{msg}(m1)$ (since bl^{i_j} is a phony block).

Proof of Claim 5-3 Let π_l be the operation in which u receives message $m2$. To complete the proof we show that (1) $i_j < l < i_{j+1}$, and (2) $potential(\mathrm{bl}^{l+1}) \leq potential(\mathrm{bl}^l) - 1$.

1. By the definition of π_l, clearly $i_j < l$. Since $u \notin \mathrm{bl}^l$, it is clear that $l < j{+}1$. Assume to the contrary that $u \in \mathrm{bl}^i$, then by Claim 1 $\mathsf{selected}_u$ equals the body of the last message received from v. Since $m1$ is the last message u received from v, $\mathsf{selected}_v = next(zero_v, \mathsf{out\_port}_v(u))$. This contradicts the fact that bl^l is a phony block.

2. Assume, to the contrary, that $potential(\mathrm{bl}^{l+1}) > potential(\mathrm{bl}^{l}) - 1$. Hence in π_l after receiving $m2$, u sends another strong message that belongs to bl^{l+1} and u performs lines 3 and 4 or 10–12. Let us consider the two cases:

   (a) If line 4 is performed, then $\mathsf{in\_port}_u(v) = parent^l_u$, and $\mathsf{msg}(m2) = \mathsf{selected}^l_u$ (line 3); but by Claim 1, $\mathsf{selected}^l_u = \mathsf{msg}(m1)$, and this is a contradiction since $\mathsf{msg}(m1) \neq \mathsf{msg}(m2)$.

   (b) In case lines 8 and 9 are performed, then $\mathsf{in\_port}_u(v) \in \mathsf{prev\_ports}^l$ and $\mathsf{prev}^l_u = \mathsf{msg}(m2)$ (line 10), but by Claim 2, $\mathsf{prev}^l_u = \mathsf{msg}(m1)$, and this is a contradiction because $\mathsf{msg}(m1) \neq \mathsf{msg}(m2)$. Hence $potential(\mathrm{bl}^{l+1}) \leq potential(\mathrm{bl}^{l}) - 1$.

Proof of Claim 5 □

Proof of Lemma 4-1 **Proof of Lemma 4** Let q_s be a state after s_1, and let q_w be a state that is $nB + n + 3$ time units after q_s. We prove that at state q_w the potential of every phony block reaches zero. Assume to the contrary that in state q_w the potential of a particular block bl^w is not zero. Hence, there exists a strong message which we denote as m^w, where $m^w \in \mathrm{bl}^w$. Let $\mathrm{bl}^s\mathrm{bl}^{s+1}\ldots\mathrm{bl}^w$ be the progression of the block from state q_s to q_w, and let $m^s m^{s+1}\ldots m^w$ be the traversal of the message in the different states $q_s q_{s+1}\ldots q_w$.

Proof of Lemma 4-2 Let us look at the group of operations $\{\pi_i\}_{s\leq i\leq w-1}$. The following two observations about the remainder of m^i and the potential of bl^i are true:

1. At least $nB+n+3$ times, there is an operation π_i such that a node in bl^i receives m^i and sends m^{i+1} (cases 1 and 3 of Corollary 5). Hence at least $nB + n + 3$ times, $remainder(m^{i+1}, \mathrm{bl}^{i+1}) = remainder(m^i, sb^i) - 1$.

2. At most $nB + 1$ times, an operation π_i exists such that a node is captured by block bl^i (cases 2 and 3 of Corollary 5).

   This follows from three main points:

   (a) by Observation 6, $potential(\mathrm{bl}^{i+1}) \leq potential(\mathrm{bl}^{i})$ $byClaim5, betweeneverytwoconsecut$ such that $potential(\mathrm{bl}^{l+1}) \leq potential(\mathrm{bl}^{l}) - 1$; and

   (b) by Corollary 3, $potential(\mathrm{bl}^{s}) \leq nB$.

By Observation 4, $remainder(m^s, \mathrm{bl}^s) \leq n+1$. $remainder(m^w, \mathrm{bl}^w) = remainder(m^s, \mathrm{bl}^s) + \Sigma_s^{w-1}(remainder(m^{i+1}, \mathrm{bl}^{i+1}) - remainder(m^i, \mathrm{bl}^i)) \leq n+1+(-1)*(nB+n+3)+(+1)*(nB+1) = -1 < 0$. This is a contradiction, since by Observation 4 $remainder(m^w, \mathrm{bl}^w) \geq 0$.

Proof of Lemma 4 □

6.2-11 Let s_2 be the state where the potential of every phony block equals zero. Let q_m be a state, after state s_2, and let q_p be a state that is two time units after q_m. Let $\mathrm{bl}^m \mathrm{bl}^{m+1} \ldots \mathrm{bl}^p = \mathrm{bl}$ be the progression of a phony block bl from state q_m to state q_p.

Lemma 5 *If the number of nodes in bl^p is k, then the number of nodes in bl^m at state q^m is at least $k+1$.*

Proof of Lemma 5-1 **Proof of Lemma 5** Let v be the tail node in bl^m at state q_m. Clearly, $\mathsf{parent}_v \neq \mathrm{nil}$, since otherwise $\mathsf{selected}_v = zero_v$ (by line 1), which is a contradiction to the fact that v belongs to a phony block. Let $\mathsf{parent}_v = \mathsf{in\_port}_v(u)$, where u is an outgoing neighbor of v.

Proof of Lemma 5-2 To prove the lemma, we show that at a maximum of two time units after q_m there exists an operation such that $\mathsf{selected}_v$ changes. This is sufficient, since no new nodes can be captured by any phony block after state s_2.

Proof of Lemma 5-3 Let p be the previous element to the tail of block bl^m at q_m. By property 6 of the block definition, $p \cup \mathrm{bl}^m$ is not a block.

Proof of Lemma 5-4 There are two cases to consider: (1) p is a message m that is transfered from u to v, or (2) p is node u.

Proof of Lemma 5-5 In case p is a message, then after a maximum of one time unit the first node in the block receives m from its parent such that $\mathsf{msg}(m) \neq \mathsf{selected}_v$ (since m cannot be added to the block) and $\mathsf{selected}_v$ changes to $zero_v$ by lines 6 and 7.

Proof of Lemma 5-6 In case p is a node, then after at most one time unit from q_m, u sends a message m by time-out. Message m cannot be added to the phony block, and after one time unit node u still cannot be added to the block (no new phony nodes can be captured after s_2). Hence, we are again in case 1.

Proof of Lemma 5 □

Lemma 6 *In $O(n)$ time units after s_2, there are no phony blocks.*

Proof of Lemma 6-1 **Proof of Lemma 6** Within $O(n)$ time units after s_2, the block contains only a weak message sequence (no nodes at all). This follows from the following facts:

- by Lemma 5, in each $O(1)$ time units after state q_m, the number of nodes in a block is reduced by at least one;
- there are at most n nodes in a phony block to start with; and
- no new nodes can be added to a phony block after state s_2 (since the potential of a phony block equals zero).

Proof of Lemma 6-2 Within $O(1)$ time units after that state, the weak-message sequence also disappears. This follows from the following statements: (1) weak messages can be sent only in a time-out; and (2) there is no node in the phony block to send them.

Proof of Lemma 6 $\square$

Let s_3 be the global state in which there are no phony blocks.

Definition 16 $\mathsf{Selected}_v^i$, *the* $\mathsf{selected}$ *value of node* v *at state* q_i, *is a derivative of* $zero_u$ *if there is a* u*-to-*v *path,* $\{v_1 v_2 \dots v_k\}$, *where* $u = v_1$ *and* $v = v_k$, *and an assignment to the variables of all the nodes along the path that defines a block on the path subject to* $\mathsf{selected}_u = zero_u$ *and* $\mathsf{selected}_{v_{j+1}}^i = \mathsf{next}(\mathsf{selected}_{v_j}^i, \mathsf{out\_port}_{v_j}(v_{j+1}))$, *where* $0 \leq j \leq k-1$. *The path is called the* derivation path from u to v.

Observation 7 *The selected value of any node in a state after* s_3 *is a derivative of some zero value of a node in the network.*

The observation follows, since there are no phony blocks in the network.

Definition 17 (Stabilization Properties)
Each framework that is suitable for our generic algorithm defines a set of legal global and final states such that the following properties hold:

1. *Each node has a stable and legal* $\mathsf{selected}$ *value. Node* v *has a legal* $\mathsf{selected}$ *value at the state after* s_3, *if its derivation path,* $\{v_0, v_1, \dots, v_k, v\}$, *denoted by the legal derivation path of* v, *satisfies the following properties:*

(a) *There is a unique node denoted r, the root of the tree, such that $v = r$ and r's legal* selected *value is* $zero_r$. *Otherwise, if $v \neq r$,* $\mathsf{selected}_v$, *the legal value of node v, is a derivative of* $zero_r$, *i.e.,* $v_0 = r$.

(b) *For every* $0 \leq i \leq k$, $\{v_0, v_1, \ldots, v_i\}$ *is a legal derivation path for a legal* $\mathsf{selected}_{v_i}$ *value.*

(c) *A legal* selected *value of node v is selected as* $\mathsf{selected}_v$ *over any other suggestion for a nonlegal* selected *value which is a derivative of some zero value of a node in the network.*

2. *Each node has a stable parent. The parent of r is nil, and the parent of every other node is a pointer to one of its incoming neighbors. The graph induced by the parent relationship is a tree.*

Definition 18 *A block is a legal derivation block of v laid over the derivation path* dp, *if the block is on the path dp and the assignment of the* selected *values of all the nodes along dp is legal.*

Remark 4 *A node can have more than one legal* selected *value; hence there can be more than one legal derivation path and legal derivation block.*

Observation 8 *If v has a legal* selected *value after s_3, then any block that contains v must be a block or a subblock of a legal derivation block.*

6.2-12 The observation follows from property (b).

Observation 9 *Let bl be a block or a subblock of a legal derivation block. Then the source of bl is a block or subblock of a legal derivation block.*

Lemma 7 *In any state q_i which is $O(n)$ or more time units after s_3, if* $\mathsf{selected}_v$ *is legal, then there exists a legal derivation block based on the derivation path from r to v.*

6.2-13 Before proving the lemma, let us define:

Definition 19 *A block without power supply is a block whose tail's* selected *value is not a* zero *value.*

Observation 10 *A legal derivation block is a block with power supply.*

6.2-13 The proof of the lemma is based on the following two claims:

Claim 6 *Let bl be a block without power supply at a state after s_3, then after $O(n)$ time units the block disappears.*

Proof of Claim 6 The proof of the claim is similar to the proof of Lemma 6, and will not be repeated here. The intuition is that a phony block is also a block without power supply. The same properties hold for a phony block that disappears and a block without power supply.

Proof of Claim 6 □

Consider a state q_i after s_3 in which $\mathsf{selected}_u$ is a legal value, and a legal derivation block is laid over *ldp*, the legal derivation path from r to u. Then the following claim applies.

Claim 7 *At any state after q_i, u has a legal derivation block which is laid over ldp.*

Proof of Claim 7-1 **Proof of Claim 7** Assume, to the contrary, that there is an operation π_w such that at state q_w, after q_i, there is a legal derivation block laid over ldp, and in state q_{w+1} after this operation there is no legal derivation block on ldp.

Proof of Claim 7-2 Hence, some node from ldp changes its state in operation π_w. Let v be that node. By property (b) of a legal derivation path, v has a legal $\mathsf{selected}$ value. By property (c), it cannot be that v changes its state in π_w: contradiction.

Proof of Claim 7 □

Proof of Lemma 7-1 **Proof of Lemma 7** Assume, to the contrary, that there exists a node v in state q_i, $O(n)$ after s_3 such that $\mathsf{selected}_v$ is legal, but there is no legal derivation block. Let bl′ be a block that contains v. Block bl′ must be a subblock of a legal derivation block at state q_i (by Observation 8); hence by Observation 10, bl′ is a block without power supply.

Proof of Lemma 7-2 By Claim 6, the source of bl′ at state q_m before s_3 is a block with power supply. Therefore, there is an operation after s_3, denoted by π_p, because of which the block is without power supply. But this is impossible, since at state q_p the source of bl′ is a legal derivation block (by Observation 9) and by Claim 7, the source of bl′ at state q_{p+1} is also a legal derivation block; hence a block with power supply (by Observation 10). This is a contradiction; hence the lemma.

Proof of Lemma 7 □

Lemma 8 *In $O(n)$ time units after s_h, the network enters state s_4 and a legal final state L such that in any state after s_4 the network is in state L. In the legal final state L, a rooted tree (as required) spans the network.*

Proof of Lemma 8-1 **Proof of Lemma 8** The proof is based on an inductive assumption that at state q_k (which is at least $2k$ time units after s_h), every node that has at least one legal derivation path that contains k nodes in its selected value is legal.

Proof of Lemma 8-2 *Base*: For $k = 1$, we should prove that the inductive assumption holds for r. By property (a) of legal derivation path, $zero_r$ is selected as a legal value of $\mathsf{selected}_r$ from all the derivative selected values for node r. Hence by line 2 and by Observation 7 the result holds.

Proof of Lemma 8-3 *Step*: Assuming the inductive assumption holds for k, then we prove it for $k+1$. Let v be a node that has a legal derivation path with $k+1$ nodes. By property (b) of a legal derivation block, node v has a neighbor w with a legal derivation path with k nodes, and hence the inductive assumption holds for node w. Therefore, by Lemma 7, a legal derivation block exists for node w at state q_k which is at least $2k$ time units after s_h. Node w receives a strong message in every time unit from its parent in the legal derivation block. This follows from the fact that the tail of a legal derivation block is r and $\mathsf{selected}_r = zero_r$ (by property a of a legal derivation block), and r sends a strong message on all of its outgoing links every time unit (by procedure time-out line 30). Hence at q_{k+1}, the state that is at least $2(k+1)$ time units after s_h, v receives two strong messages from w. The body part of these messages is a suggestion for a legal selected. If those suggestions are selected as $\mathsf{selected}_v$, then upon receiving the first message, prev is updated (lines 14–21), and upon receiving the second message, selected is updated (lines 11–13). Therefore, at q_{k+1}, $\mathsf{selected}_v$ has a legal value. Otherwise, this suggestion for $\mathsf{selected}_v$ is not selected as $\mathsf{selected}_v$. By property c of a legal derivation block, the latter can happen only if $\mathsf{selected}_v$ already has a legal *selected* value. Hence the lemma holds.

Proof of Lemma 8 □

6.2-14 This completes the proof of the main theorem.

Lemma 9 *The time complexity of the BFS is $O(D)$.*

Proof of Lemma 9 The same inductive assumption as in Lemma 8 holds, with some changes when the starting point is a state after s_1 and not after s_3.

Proof of Lemma 9 □

Remark 5 *The proof ignored a remark that was made about the model in Sections 2 and 4, thus assuming that the rate at which messages are processed is faster than the rate at which links transmit them. It is easy to modify the proof to hold with the remark. In effect, the remark is equivalent to losing some messages on the links which may only decrease the potential of phony blocks, and does not disturb the power-supply principle. Notice that in the remark we introduce two buffers at each port, thus preventing the possibility of a few blocks uniting into one block, i.e., not affecting the correctness of Lemma 3.*

6.3 Conclusions

6.3-1 A simple method for designing self-stabilizing algorithms for unidirectional and bi-directional networks has been presented in this paper. The key advantages of our methodology are its simplicity, it works in arbitrary synchronous and asynchronous unidirectional networks, and it does not require any a priori knowledge about the network topology or size. The algorithms presented here stabilize in *O(n)* time units, and any fake message disappears in time that is proportional to the number of the fake messages in the greatest phony block. To demonstrate the capabilities of our methodology, we present algorithms for constructing DFS and BFS trees. These trees are up-trees, that is, there exists a path in the tree from the root to any node.

6.3-2 An open question is to design an algorithm under similar assumptions and complexities but for the construction of a down-tree, that is, a tree in which there is a unidirectional path from any node to the root. The combination of up-tree and down-tree algorithms would facilitate the automatic translation of any bidirectional distributed algorithm into a unidirectional self-stabilizing version of the algorithm.

Appendix: The Lower Bound of the Synchronous Algorithm

A-1 In Figure 6, a simple scenario in which our algorithm has $\Omega(n)$ time complexity is presented. In the figure, beside each node we have placed the record [leader, distance from the leader], plus the node id in bold-face type. Let us look at the block of size $n-1$ with a fake id of 1. To simplify, we describe the scenario under the synchronous model. In every round, only one node from the block with fake id 1 disappears; this is the node with the smallest distance in the block.

A-2 For example, in the first round, the node with an id of 80 changes its state, since it receives only one message [10, 0] from node 10, and according to this message its leader should be 10 and not 1. Assuming this is the first time node 80 receives the change, the node changes its state to [80, 0]; otherwise, it changes to [10, 1]. All other nodes in the block with fake id 1 do not change their state, because they get a message that reassures their state from their incoming neigbor in the block of fake id 1. Similarly, in the next round, node 70 changes its state, since it receives the messages [80, 0] or [10, 1] from node 80 and [10, 0] from node 10, which do not support its current leader, 1. All other nodes in the block with a fake id of 1 do not change their states because they get messages that reassure their states from their incoming neigbors in the block with fake id of 1. Hence, after only $n-1$ rounds, the block with a fake id of 1 disappears, even though the diameter of the network is 2.

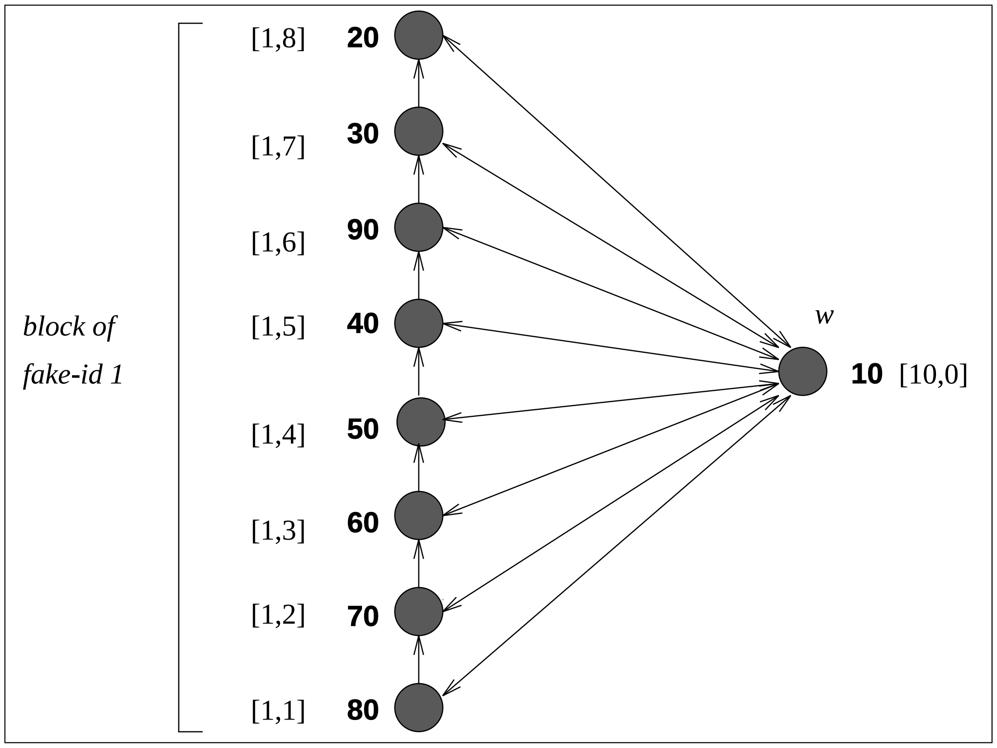

Figure 6: An example

Acknowledgment of support: We would like to thank Shlomi Dolev for several helpful discussions. This work was supported by the Broadband Telecommunications R&D Consortium, administered by the chief scientist of the Israeli Ministry of Industry and Trade.

References

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proceedings of the 28th IEEE Annual Symposium on Foundation of Computer Science*, pages 358–370, Los Alamitos, CA, October 1987. IEEE.

[AB93] Y. Afek and G. M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing Journal*, 7:27–34, 1993. Also abstracted in *Proceedings of the 8th IEEE Symposium on Reliable Distributed Systems*, Los Alamitos, CA, pages 10–12 1989. IEEE.

[AEYH92] E. Anagnostou, R. El-Yaniv, and V. Hadzilacos. Memory adaptive self-stabilizing protocols. In *Proceedings of the 6th International Workshop on Distributed Algorithms*, volume 647 of *Lecture Notes in Computer Science*, pages 203–220, Berlin, November 1992. Springer-Verlag.

[AG94a] Y. Afek and E. Gafni. Distributed algorithms for unidirectional networks. *Siam Journal on Computing*, 23(6):1152–1178, December 1994.

[AG94b] A. Arora and M. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43:1026–1038, 1994.

[AKM$^+$93] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self stabilizing synchronization. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 652–661, New York, May 1993. ACM.

[AKY90] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self stabilizing protocols for general networks. In *Proceedings of the 4th International Workshop on Distributed Algorithms*, volume

484 of *Lecture Notes in Computer Science*, pages 15–28, Berlin, September 1990. Springer-Verlag.

[ALss] Y. Afek and T. Lev. Distributed synchronization protocols for SDH networks. Submitted for publication, November 1995, In Press.

[APSV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proceedings of the 32nd IEEE Annual Symposium on Foundations of Computer Science*, pages 268–277, Los Alamitos, CA, October 1991. IEEE.

[APSVD94] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-stabilizing by local checking and global reset. In *International Workshop on Distributed Algorithms*, pages 326–339, Berlin, 1994. Springer-Verlag.

[AV91] B. Awerbuch and G. Varghese. Distributed program checking: A paradigm for building self-stabilizing distributed protocols. In *Proceedings of the 32nd IEEE Annual Symposium on Foundation of Computer Science*, pages 258–267, Los Alamitos, CA, October 1991. IEEE.

[BGM90] J. E. Burns, M. G. Gouda, and R. E. Miller. Stabilization and pseudo stabilization. Technical Report TR-90-13, The University of Texas at Austin, May 1990.

[BGW89] G. M. Brown, M. G. Gouda, and C.-L. Wu. Token systems that self-stabilize. *IEEE Transactions on Computers*, c38(6):845–852, 1989.

[BP89] J. Burns and J. Pachl. Uniform self-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.

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

[DIM91] S. Dolev, A. Israeli, and S. Moran. Resource bounds for self stabilizing message driven protocols. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 281–294, New York, August 1991. ACM.

[DIM94] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing Journal*, 7, 1994. Also in *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, New York, August 1990. ACM.

[DKR82] D. Dolev, M. Klawe, and M. Rodeh. An $O(n \log n)$ unidirectional algorithm for extrema finding in a circle. *Journal of Algorithm*, 3:245–260, 1982.

[ELW90] S. Even, A. Litman, and P. Winkler. Computing with snakes in directed networks of automata. In *Proceedings of the 31st IEEE Annual Symposium on Foundations of Computer Science*, pages 740–745, Los Alamitos, CA, October 1990. IEEE.

[GA84] E. Gafni and Y. Afek. Election and traversal in unidirectional networks. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 190–198, New York, August 1984. ACM.

[GK84] E. Gafni and W. Korfhage. Distributed election in unidirectional Eulerian networks. In *Proceedings of the 22nd Annual Allerton Conference on Communication, Control, and Computing*, October 1984.

[GKA83] M. Gerla, L. Kleinrock, and Y. Afek. A distributed routing algorithm for unidirectional networks. In *Proceedings of the IEEE Global Telecommunications Conference, GLOBCOM '83*, pages 19.3.1–19.3.5, Los Alamitos, CA, 1983. IEEE.

[Her90] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.

[IJ93] A. Israeli and M. Jalfon. Uniform self-stabilizing ring orientation. *Information and Computation*, 104:175–196, 1993.

[KP90] S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. In M. Evangelist and S. Katz, editors, *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 91–101, New York, August 1990. ACM.

[Kut88] S. Kutten. Stepwise construction of an efficient distributed traversing algorithm for general strongly connected directed networks. In J. Raviv, editor, *Proceedings of the 9th International Conference on Computer Communication*, pages 446–452, Amsterdam, October 1988. Elsevier.

[MOOY92] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant space. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 667–678, New York, May 1992. ACM.

[MOY96] A. Mayer, R. Ostrovsky, and M. Yung. Self-stabilizing algorithms for synchronous uni-directional rings. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–573, New York, January 1996. ACM.

[Mul88] N. Multari. Self-stabilizing protocols. PhD Thesis, Department of Computer Sciences, University of Texas, 1988.

[OW95] Rafail Ostrovsky and Daniel Wilkerson. Faster computation on directed networks of automata. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 38–46, New York, August 1995. ACM.

[Pet82] G. L. Peterson. An $O(nlogn)$ unidirectional algorithm for the circular extrema problem. *ACM Transactions on Programming and Language Systems*, 4(4):758–762, October 1982.

[Taj77] W. P. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Communications of the ACM*, 20-7:477–485, 1977.