Time Optimal Self-Stabilizing Spanning Tree Algorithms

by

Sudhanshu Madan Aggarwal

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science

and

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

© Sudhanshu Madan Aggarwal, MCMXCIV. All rights reserved.

Author...
Department of Electrical Engineering and Computer Science
May 20, 1994

Certified by..
Dr. Shay Kutten
Manager – Distributed Computing, IBM T. J. Watson Research Center
Thesis Supervisor

Certified by..
Nancy A. Lynch
Professor of Computer Science
Thesis Supervisor

Certified by..
Roberto Segala
Research Associate, MIT Laboratory for Computer Science
Thesis Supervisor

Accepted by..
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Time Optimal Self-Stabilizing Spanning Tree Algorithms

by

Sudhanshu Madan Aggarwal

Submitted to the Department of Electrical Engineering and Computer Science

on May 20, 1994, in partial fulfillment of the

requirements for the degrees of

Bachelor of Science

and

Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis presents time-optimal self-stabilizing algorithms for distributed spanning tree computation in asynchronous networks. We present both a randomized algorithm for anonymous networks as well as a deterministic version for ID-based networks. Our protocols are the first to be time-optimal (i.e. stabilize in time $O(diameter)$) without any prior knowledge of the network size or diameter. Both results are achieved through a technique of symmetry breaking that may be of independent interest.

Executions of randomized distributed algorithms contain a combination of nondeterministic and probabilistic choices; these choices often involve subtle interactions that often make such algorithms difficult to verify and analyze. Segala and Lynch have recently developed the *Probabilistic Automata* model to aid in reasoning about randomized distributed algorithms; their model is related to the earlier work of Lynch and Vaandrager. We use the Probabilistic Automata formalism to analyze the correctness and time complexity of our randomized algorithm for anonymous networks; in doing so, we demonstrate the effectiveness of the formalism in reasoning about randomized algorithms.

Thesis Supervisor: Dr. Shay Kutten
Title: Manager-Distributed Computing, IBM T.J.Watson Research Center

Thesis Supervisor: Nancy A. Lynch
Title: Professor of Computer Science

Thesis Supervisor: Roberto Segala
Title: Research Associate, MIT Laboratory for Computer Science

Acknowledgments

I would like to thank my mentors and colleagues who have guided me through the course of this long project.

I would like to thank Shay Kutten for suggesting this research topic in the first place. Shay has been instrumental in guiding me through the earlier phases of this work; he has also been a most effective mentor and friend. I also thank IBM for going to great lengths to provide me with an opportunity to work with Shay.

I would like to thank Nancy Lynch for her exemplary patience and encouragement throughout the course of this work. Through Nancy I have been exposed to the notions of scholarship and rigor; I am also grateful for her willingness to read numerous drafts at long intervals.

This thesis owes much to Roberto Segala. Roberto provided the motivation for formalizing the randomized algorithm; he also suggested the state set $\mathcal{C}^1$ which plays a crucial role in the proof. Roberto also "showed me the ropes" around LCS.

I would also like to thank Boaz Patt-Shamir, Amir Herzberg, Baruch Awerbuch and Alain Mayer for the helpful discussions I have had with them in connection with this work. It was Boaz who first suggested using the Afek-Matias probability distribution, which provided a basis for attacking the initial problem.

Finally, I would like to thank my parents, without whose sacrifice and love this thesis would not have been possible.

Contents

List of Figures

Chapter 1

Introduction

The task of *spanning tree construction* is a basic primitive in communication networks. Many crucial network tasks, such as network reset (and thus any input/output task), leader election, broadcast, topology update, and distributed database maintenance, can be efficiently carried out in the presence of a tree defined on the network nodes spanning the entire network. Improving the efficiency of the underlying spanning tree algorithm usually also correspondingly improves the efficiency of the particular task at hand.

In practice, computation in asynchronous distributed networks is made much more difficult because of the possibility of numerous kinds of *faults*. Nodes may crash or get corrupted; links may fail or deliver erroneous messages. Further, nodes or links may enter or leave the network at any time. A very important concept in the context of this problem is that of *self-stabilization*, first introduced by Dijkstra [Dij74]. Self-stabilization implies the ability of the system to recover from *any* transient fault that changes the state of the system. Dijkstra gave the example of a token-ring network which is always supposed to have exactly one token. If, through some error, the network were to have zero or two tokens, a self-stabilizing token ring protocol would be able to automatically recover or "stabilize" to a state where the network has exactly one token.

More precisely, *a self-stabilizing algorithm on a system $\mathcal{S}$ (e.g. the network) reaching a set*

of legal states $\mathcal{P}$ is eventually able to bring $\mathcal{S}$ to a state in $\mathcal{P}$ when started in *any arbitrary initial state*. In Dijkstra's token-ring example, $\mathcal{P}$ is the set of states in which the ring has exactly one token. For a self-stabilizing spanning tree algorithm, $\mathcal{P}$ would be the set of states having a spanning tree defined on the network nodes. As we can consider the state of the system after a transient error to be an arbitrary state, a self-stabilizing system will eventually "recover" from any non-repeating error. Thus self-stabilization is a very strong and highly desirable fault-tolerance property.

We would therefore like to have an *efficient self-stabilizing algorithm for spanning tree construction in asynchronous networks*.

A key measure of efficiency is the *stabilization time*, which is the maximum time taken for the algorithm to converge to a "spanning tree" state, starting from an arbitrary state. Let δ be the *diameter* of the network, and let n be the *network size* – the number of nodes in the network. Then that the optimal stabilization time must necessarily be $\Omega(\delta)$.

Several factors influence the "difficulty" of the protocol. The protocol can be designed for networks that are either *ID-based* (each node has a unique "hard-wired" ID), or for networks that are *anonymous* (in which nodes lack unique IDs, so there is no *a priori* way of distinguishing them). The protocol may either "know" the network size n, or it may "know" some upper bound on n, or it may "know" nothing whatsoever. Similarly, it may or may not "know" in advance a bound on the diameter δ. Of course, the more "knowledge" a protocol "is given" about the network, the easier it becomes to achieve its objectives.

Previous Work

Following the pioneering work of [Dij74], there has been considerable work in this area. [Ang 80] showed that no deterministic algorithm can construct a spanning tree in an anonymous symmetric network. [AKY90] gave an ID-based self-stabilizing spanning tree protocol with a stabilization time of $O(n^2)$ and a randomized protocol for anonymous networks that runs in $O(n \log n)$ time. They presented the technique of "local checking" and "local detection," used in

many subsequent papers. [AG90] gave an ID-based self-stabilizing spanning tree protocol with time complexity $O(N^2)$, where N is a pre-specified *bound* on the network size n. [APV91] gave an ID-based self-stabilizing spanning tree protocol (based on a *reset* protocol) that stabilizes in $O(n)$ time.

[DIM91] gave a self-stabilizing spanning tree algorithm for anonymous networks that runs in expected $O(\delta \log n)$ time. [AM89] gave a Monte-Carlo spanning tree protocol for anonymous networks that works in $O(\delta)$ time; however, their protocol is *not* self-stabilizing. (A Monte-Carlo algorithm terminates in bounded time but succeeds with probability $p < 1$; a Las-Vegas algorithm may not terminate in bounded time but always succeeds.) With the exception of [AG90], all the other works mentioned above do not assume any prior knowledge of the network size n or the diameter δ.

[DIM91] also mentioned a self-stabilizing spanning tree protocol for anonymous networks that requires $O(\delta)$ time (and is thus time-optimal), but requires prior knowledge of a bound N on the network size. Recently, [AKMPV93] have developed a time-optimal self-stabilizing spanning tree protocol for ID-based networks; they, too, require prior knowledge of a bound D on the diameter of the network.

Our Results

We present the first time-optimal self-stabilizing spanning tree algorithms *that do not need any prior knowledge of the network size or diameter*. We present both a randomized Las-Vegas algorithm for anonymous networks and a deterministic version for ID-based networks. Both our protocols stabilize in expected $O(\delta)$ time.

Thus, with respect to the $O(\delta \log n)$-time protocol of [DIM91], we decrease the time complexity to $O(\delta)$, and compared to their $O(\delta)$-time protocol, we do not need a bound N on the network size. Unlike [AKMPV93], we do not need a bound D on the diameter.

Note that for random graphs, the expected diameter δ is comparable to $\log n$. For real networks, such as the Internet, the diameter is usually *less* than $\log n$. Thus, decreasing the

time complexity from $O(\delta \log n)$ (as in [DIM91]) to $O(\delta)$ represents an improvement in the time required to less than the square root of that required earlier.

Both of our protocols employ a novel technique in self-stabilization. A major concern in self-stabilizing systems has been contending with "wrong information". For example, an important problem that arises in spanning tree algorithms is the *ghost root* phenomenon—some nodes in the network may "believe" the existence of a root node that doesn't really exist. Most previous approaches to the problem have relied on costly non-local operations such as *root verification*, *network reset*, or *tree dismantling* to *eliminate* the ghost root. Our technique, on the other hand, is to *modify* incorrect information instead of perform the expensive process of *eliminating* it. (A similar idea to that of "correcting information" was implicitly used by [DIM91].) The modification is done locally but in a careful manner: local modifications of wrong information have important desirable global consequences. We do it without incurring the large overhead of global operations such as *reset* etc. Compared to [DIM91], we do stronger corrections (but still without causing global overhead). The stronger local corrections enable us to have a better running time.

Chapter 2

The Model

We assume that the network is represented by an undirected graph $G = (V, E)$; G consists of a set of *processors* denoted by $V = \{v_1, v_2, \ldots, v_n\}$ and a set of *links* denoted by $E = \{E_1, E_2, \ldots\}$ where each $E_i \in E = (v_j, v_k)$ for some j, k. In an *ID-based* network, each processor is assigned a unique ID that is "hard-wired" in its memory. In an *anonymous* or *uniform* network, all processors of the same degree are identical; they do not have unique IDs assigned to them. We refer to the number of processors n as the *size* of the network. The *distance* between any two processors u and v is the lowest number of links on any path connecting u and v in G. (In an anonymous network, the labels u and v are used for convenience—they are not the IDs of the nodes referred to.) The *diameter* of the network is the maximum distance between any two nodes in V; we denote the diameter by δ. The set of neighbors of node u, denoted $Nbrs(u)$, is the set $\{v \in V \mid (u, v) \in E\}$.

The *degree* of a node v is the number of links incident upon node v. We assume that each processor maintains a total order on its neighbors.

The network is *asynchronous*; processors perform computation steps independently of each other and at arbitrary rates.

We assume that processors communicate by *shared memory*. In the shared memory model, each processor is associated with a set of *registers*, possibly partitioned into a set of *local*

registers and a set of *shared registers*. Processors communicate by performing write operations on their registers and read operations on the shared registers of their neighbors. All reads and writes are *atomic*—reads/writes behave as though they occur instantaneously.

A network communicating through shared memory, as described above, can be modeled as a *probabilistic automaton* ([SL94], [LSS94]).

2.1 Probabilistic Automata

In this section we give only a simplified version of the model of [SL94] which is sufficient for our purposes.

2.1.1 Automata

Definition 2.1 A *probabilistic automaton M* consists of four components:

- a set *states*(M) of states.

- a nonempty set *start*(M) $\subseteq$ *states*(M) of start states.

- a set *acts*(M) of actions.

- a transition relation *steps*(M) $\subseteq$ *states*(M) $\times$ *acts*(M) $\times$ *Probs*(*states*(M)), where the set *Probs*(*states*(M)) is the set of probability spaces (Ω, Σ, P) such that $\Omega \subseteq$ *states*(M) and $\Sigma = 2^{\Omega}$. ∎

Thus, a probabilistic automaton is a state machine with a labeled transition relation such that the state reached during a step is determined by some probability distribution. For example, the process of choosing a random color from $\{0, 1, 2\}$ is represented by a step labeled with an action NEXT-COLOR where the next state contains the random color choice and is determined by a probability distribution over the three possible outcomes. A probabilistic

18

automaton also allows nondeterministic choices over steps. A key instance of nondeterminism is the choice of which processor in a network takes the next step.

Given a state s, let $\mathcal{D}(s)$, the Dirac distribution on s, denote the probability space that assigns probability 1 to s. Specifically, $\mathcal{D}(s) = (\{s\}, 2^{\{s\}}, P)$ such that $P[\{s\}] = 1$. As a notational convention we write $(s, a, s') \in steps(M)$ whenever $(s, a, \mathcal{D}(s')) \in steps(M)$.

2.1.2   Executions

An *execution fragment* α of a probabilistic automaton M is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution fragment is finite, ending in a state; $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$, where for each i there exists a probability space (Ω, Σ, P) such that $(s_i, a_{i+1}, (\Omega, \Sigma, P)) \in steps(M)$ and $s_{i+1} \in \Omega$. Iff $i < j$, we say "s_i precedes s_j in α," or "s_j follows s_i in α." Denote by $fstate(\alpha)$ the first state of α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. Furthermore, denote by $frag^*(M)$ and $frag(M)$ the sets of finite and all execution fragments of M, respectively. An *execution* is an execution fragment whose first state is a start state. Denote by $exec^*(M)$ and $exec(M)$ the sets of finite and all executions of M, respectively. A state s of M is *reachable* if there exists a finite execution of M that ends in s. Denote by $rstates(M)$ the set of reachable states of M.

A finite execution fragment $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of M and an execution fragment $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of M can be *concatenated*. In this case the concatenation, written $\alpha_1 ^\frown \alpha_2$, is the execution fragment $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. An execution fragment α_1 of M is a *prefix* of an execution fragment α_2 of M, written $\alpha_1 \le \alpha_2$, if either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution fragment α_1' of M such that $\alpha_2 = \alpha_1 ^\frown \alpha_1'$. If $\alpha = \alpha_1 ^\frown \alpha_2$, then we denote α_2 with $\alpha \triangleright \alpha_1$ (read α after α_1).

Let U be a subset of $states(M)$. Set U is *closed*, written $U \longrightarrow U_{\boxdot}$, if for any $s \in U$ and any step $(s, a, (\Omega, \Sigma, P))$, $\Omega \subseteq U$. Thus if $U \longrightarrow U_{\boxdot}$, once an execution reaches a state in U, it remains in U. We say that an execution fragment α is in U if every state in α is in U.

2.1.3 Adversaries

In order to study the probabilistic behavior of a probabilistic automaton, some mechanism to remove nondeterminism is necessary. The mechanism that removes the nondeterminism can be viewed as an *adversary*. In distributed systems the adversary is often called the *scheduler*, because its main job may be to decide which process should take the next step.

Definition 2.2 An *adversary* for a probabilistic automaton M is a function $\mathcal{A}$ taking a finite execution fragment of M and giving back either nothing or one of the enabled steps of M if there are any. Denote the set of adversaries for M by $Advs_M$. ∎

2.1.4 Execution Automata

Once an adversary is chosen, a probabilistic automaton can run under the control of the chosen adversary. The result of the interaction is called an execution automaton. Note that there are no nondeterministic choices left in an execution automaton.

Definition 2.3 An *execution automaton* H of a probabilistic automaton M is a fully probabilistic automaton such that

1. $states(H) \subseteq frag^*(M)$.

2. for each step $(\alpha, a, (\Omega, \Sigma, P))$ of H there is a step $(lstate(\alpha), a, (\Omega', \Sigma', P'))$ of M, called the corresponding step, such that $\Omega = \{\alpha a s | s \in \Omega'\}$ and $P[\alpha a s] = P'[s]$ for each $s \in \Omega'$.

3. each state of H is reachable, i.e., for each $\alpha \in states(H)$ there exists an execution of H leading to state α. ∎

Definition 2.4 Given a probabilistic automaton M, an adversary $\mathcal{A} \in Advs_M$, and an execution fragment $\alpha \in frag^*(M)$, the execution $H(M, \mathcal{A}, \alpha)$ of M under adversary $\mathcal{A}$ with starting fragment α is the execution automaton of M whose start state is α and such that for each step $(\alpha', a, (\Omega, \Sigma, P)) \in steps(H(M, \mathcal{A}, \alpha))$, its corresponding step is the step $\mathcal{A}(\alpha')$. ∎

To ease the notation, we define an operator $\alpha\uparrow$ that takes an execution of M and gives back the corresponding execution of H, and $\alpha\downarrow$ that takes an execution of H and gives back the corresponding execution of M.

2.1.5 Events

Given an execution automaton H, an event is expressed by means of a set of maximal executions of H, where a maximal execution of H is either infinite, or it is finite and its last state does not enable any step in H. For example, the event "eventually action a occurs" is the set of maximal executions of H where action a does occur. A more formal definition follows. The sample space Ω_H is the set of maximal executions of H. The σ-algebra Σ_H is the smallest σ-algebra that contains the set of *rectangles* R_α, consisting of the executions of Ω_H having α as a prefix. The probability measure P_H is the unique extension of the probability measure defined on rectangles as follows: $P_H[R_\alpha]$ is the product of the probabilities of each step of H generating α.

Definition 2.5 An event schema e for a probabilistic automaton M is a function associating an event of Σ_H with each execution automaton H of M. ∎

2.1.6 Timing

To mark the passage of time, we include in each state s a real component $s.now$, and include a special time passage action ν in $acts(M)$, which increments $s.now$. For all $s \in start(M)$, $s.now = 0$.

Definition 2.6 (Duration of an execution fragment) The *duration* of an execution fragment α is defined as $(lstate(\alpha).now - fstate(\alpha).now)$.

A statement of the form "within time t in execution α, property P holds" means that property P holds for some state s in α such that $s.now \leq fstate(\alpha).now + t$. The statement

"after time t, property P holds" implies that property P holds for all states s in α such that $s.now > fstate(\alpha).now + t$.

2.1.7   Adversary Schemas

We close this section with one final definition. The time bound for our randomized protocol states that starting from any state, no matter how the steps of the system are scheduled, the network forms a spanning tree within expected $O(diameter)$ time. However, this claim can only be valid if the adversary is fair (as defined above). Thus, we need a way to restrict the set of adversaries for a probabilistic automaton. The following definition provides a general way of doing this.

Definition 2.7 An *adversary schema* for a probabilistic automaton M, denoted by $Advs$, is a subset of $Advs_M$. ■

2.2   Composability

In this section, we introduce a key theorem of [SL94], the *composability theorem*.

The statement $U \xrightarrow[p]{t}_{Advs} U'$ means that, starting from any state of U and under any adversary $\mathcal{A}$ of $Advs$, the probability of reaching a state of U' within time t is at least p. The suffix $Advs$ is omitted whenever we think it is clear from the context.

Definition 2.8 Let $e_{U',t}$ be the event schema that, applied to an execution automaton H, returns the set of maximal executions α of H where a state from U' is reached in some state of α within time t. Then $U \xrightarrow[p]{t}_{Advs} U'$ iff for each $s \in U$ and each $\mathcal{A} \in Advs$, $P_{H(M,\mathcal{A},s)}[e_{U',t}(H(M,\mathcal{A},s))] \geq p$. ■

Proposition 2.9 *Let U, U', U'' be sets of states of a probabilistic automaton M.*
If $U \xrightarrow[p]{t} U'$, then $U \cup U'' \xrightarrow[p]{t} U' \cup U''$. ■

In order to compose time bound statements, we need a restriction for adversary schemas stating that the power of the adversary schema is not reduced if a prefix of the past history of the execution is not known. Most adversary schemas that appear in the literature satisfy this restriction.

Definition 2.10 An adversary schema $Advs$ for a probabilistic automaton M is *execution closed* if, for each $\mathcal{A} \in Advs$ and each finite execution fragment $\alpha \in frag^*(M)$, there exists an adversary $\mathcal{A}' \in Advs$ such that for each execution fragment $\alpha' \in frag^*(M)$ with $lstate(\alpha) = fstate(\alpha')$, $\mathcal{A}'(\alpha') = \mathcal{A}(\alpha \frown \alpha')$. ∎

Theorem 2.11 (Composability theorem) *Let $Advs$ be an execution closed adversary schema for a probabilistic timed automaton M, and let U, U', U'' be sets of states of M.*
If $U \xrightarrow[p_1]{t_1}_{Advs} U'$ and $U' \xrightarrow[p_2]{t_2}_{Advs} U''$, then $U \xrightarrow[p_1 p_2]{t_1 + t_2}_{Advs} U''$. ∎

Corollary 2.12 *Let $Advs$ be an execution closed adversary schema for a probabilistic timed automaton M, and let U, U_1, U_2, ..., U_n, U^* be sets of states of M.*
If $U \xrightarrow[1]{t}_{Advs} U_1 \cup U_2, \cup \ldots \cup U_n$, and if $U_i \xrightarrow[p_i]{t_i}_{Advs} U^*$ for all i, then

$$U \xrightarrow[\min(p_1, p_2, \ldots, p_i)]{t + \max(t_1, t_2, \ldots, t_i)}_{Advs} U^*$$

∎

2.3   Networks as Probabilistic Automata

In this section we briefly describe how self-stabilizing protocols running on networks with shared-memory links can be modeled using probabilistic automata.

Self-stabilizing network protocols operate on networks that are *dynamic*—the set of processors or links may change during the execution. A change in the status of a processor or link is communicated to the processors it connects by a low level self-stabilizing protocol. Further,

the state of a processor may change arbitrarily (not by an algorithmic step, but by "memory corruption"). We assume that the sequence of topological changes and non-algorithmic state changes is finite and that eventually such events cease. This allows us to ignore topological and state changes during an execution α of our protocol, as the last such change can be considered to change the network state to an arbitrary start state s of a new change-free execution. The time complexity measures the time taken for the protocol to succeed after the last such change.

The network $G(V, E)$ can be represented by a "global" probabilistic automaton M whose state contains a vector of states of all its processors. We assume that the state $s_{[i]}$ of a processor i fully describes its internal state and the values written in all its registers. Thus the global state s contains $\{s_{[1]}, s_{[2]}, \ldots, s_{[n]}\}$; in addition, it also contains timing information (e.g. now). The local computation at each processor consists of a sequence of atomic actions; the set $acts(M)$ of actions of the global network includes the set of actions of each of its nodes, and the time passage action ν.

2.3.1 Fairness

Let $vis(M)$ denote the non-time-passage actions of $acts(M)$. For the time complexity analysis, our protocols require that each action of $vis(M)$ be executed in every unit of time. To this end, for each action a in $vis(M)$, we include in state s a (real) "deadline" for that action, $s.deadline(a)$; this deadline represents the latest time by which action a must be performed again. For all $s \in start(M)$, $s.deadline(a) = 1$. A time passage step (s, ν, s') of M must satisfy the following condition: $s'.now \leq \min_{a \in vis(M)}\{deadline(a)\}$. For a non-time-passage action (s, a, s'), $s'.now = s.now$, and $s'.deadline(a) = s.now + 1$. Note that this construction guarantees that in any execution fragment $\alpha = s_0 a_1 s_1 a_2 \ldots$ of M if $lstate(\alpha).now \geq fstate(\alpha).now + 1$, then for every action a in vis there exists a step (s, a, s') in α.

For stating time bounds, we will need to assume *fair adversaries*. $\mathcal{A}$ is said to be *fair* iff the time advances without bound in every infinite execution fragment generated by $\mathcal{A}$. (Note that this rules out "Zeno executions.") Let $Fairadvs(M)$ denote the adversary schema consisting

of fair adversaries of M. From the definitions, it can be seen that any infinite execution $\alpha = s_0 a_1 s_1 a_2 \ldots$ of M generated by a fair adversary $\mathcal{A}$ can be partitioned into an infinite number of "rounds," such that each processor performs each one of its enabled actions at least once in every round.

Also, note that the adversary schema $Fairadvs(M)$ is execution-closed (cf. Definition 2.10).

Chapter 3

General Approaches to Spanning Tree Construction

Spanning tree algorithms usually utilize variants of a common overall scheme. We first describe the basic scheme which assumes the existence of unique node IDs. Each node is associated with a "priority," which could initially be the node's ID, for instance. At any instant during the algorithm's progress, the network is logically partitioned into a spanning forest, which is defined by *parent* pointers maintained by the nodes. Initially (unless initialized by the adversary), this forest consists of the single-node trees defined by the network nodes themselves (i.e. *parent* = **nil** at all nodes, so each node is a root). Starting from this configuration, the nodes gradually coalesce into larger trees. Each node keeps track of the priority of the root of its tree. The goal is to produce a spanning tree rooted at the node with the highest priority. Nodes in the forest keep on exchanging root priorities with their neighbors. When a node u notices a neighbor v with a higher root priority, it attaches itself to v's tree by making v its parent ($parent_u \leftarrow v$). Thus, trees with higher root priorities *overrun* trees with lower ones. Since the priorities are totally ordered, eventually all nodes in the network form a single tree rooted at the node with the highest priority. This simple ID-based scheme is not self-stabilizing, since if we allow "corrupted" initial states, nodes may "believe in" a highest priority that is not

actually possessed by any root.

To adapt the ID-based scheme to an anonymous network (i.e. with no pre-assigned IDs), we need randomization to break symmetry between the processors. Each node in the network flips coins to arrive at a random ID, and participates in the tree construction process described above. Since IDs (and hence priorities) are chosen randomly, it is possible that the node with the highest priority in the network is not unique; there could be several such nodes with highest priority p. In such a situation, the above algorithm would halt when the network forms a spanning forest, with each tree rooted at one of the nodes with priority p. In this final state, all nodes would have the same ID; thus coalescing would cease at this point.

To detect such "multiple highest priorities," [AKY90] and [DIM 91] proposed the method of *recoloring* trees. In typical recoloring schemes, each tree is associated with a randomly chosen *color*. The root chooses a color at random from a small set of "colors" $\mathcal{C}$ of constant size (e.g. $\mathcal{C} = \{0, 1, 2, 3\}$). This color is propagated through the entire tree rooted at that root. When the root receives confirmation that the entire tree has been colored with its color (through a simple acknowledgement mechanism), it chooses a new color. The process is repeated forever.

If there are several neighboring trees with priority p, there must exist nodes that are linked to neighbors not in their own tree. Since tree colors are chosen randomly, neighboring nodes that belong to different trees will assume different sequences of colors over time; this fact can be exploited to let such neighbors detect their affiliation to different trees.

In the scheme proposed by [AKY90], the sequence of colors chosen by a root to color its tree is "alternating" - of the form $(c_1, c_s, c_2, c_s, c_3, c_s, \ldots)$, where c_s is a special color, "no-color," and $c_i \neq$ no-color for all i. We can represent "no-color" by the color 0; then $c_i \neq 0$ for all i. Thus when a root receives acknowledgement about its entire tree being colored with a non-zero color, it colors its tree with color 0. When its tree is entirely colored with color 0, it again recolors its tree with a non-zero color. In this scheme, if the node's own color c_i is non-zero, then if it notices a neighbor with a non-zero color different from its own color, it can correctly conclude that that neighbor belongs to a different tree. Since the scheduler is assumed to be adversarial, additional constraints are imposed on the acknowledgement mechanism; details

are presented in Section 5.

If a node v detects another tree, its root is informed of the condition. When a root learns of the existence of another tree rooted at the same ID, in the [AKY90] and [DIM91] schemes the root *extends* its ID by a randomly chosen bit and continues the protocol. Extending IDs is a way of breaking symmetry; eventually the roots in the network have appended enough random bits to their IDs so that there is a unique root with the highest ID, and subsequently a unique tree spanning the entire network.

Our technical contribution in this paper is twofold. First, we develop a framework for ID extension and generalize the concept. Our generalization enables us to reduce the time complexity of the randomized protocol to $O(d)$, without prior knowledge of the size or diameter of the network. Our second main contribution is to use the concept of extension to efficiently confer the property of *self-stabilization* upon the basic deterministic scheme for ID-based networks, thus enabling us to give the first deterministic spanning tree protocol that is time-optimal (i.e. $O(d)$ time) without prior knowledge of bounds on the network size or diameter.

Intuitively, the $\log n$ factor in the previous randomized result came from the need to initiate a new *competition* every time two trees "collided." Every time a tree T noticed another tree $\overline{T}$ with the same root ID, T would randomly extend its ID to try to "win" over $\overline{T}$. Our new method usually needs just $O(1)$ ID extensions per node to converge to a spanning tree, as opposed to $O(\log n)$ extensions in the previous scheme. To achieve this the extension needs to be done in a careful way. When several IDs are independently extended, only one extended ID ought to "win," in order to prevent the need for additional competition. Further, independent extensions must attempt to preserve existing order: they must not make a previously "beaten" tree become the maximum, since this will prevent progress by possibly necessitating new competition(s).

Previous approaches to the deterministic version attempt to form a spanning tree at the node v_l with the highest (or lowest) "hard-wired" ID. In doing so, they have to contend with the *ghost root* problem—eliminating all "belief" in the ghost root usually necessitates an "extra" $\Omega(d)$ addition to the time complexity. We exploit our intuitive results about ID extension to

modify belief in the ghost root. In our scheme, as opposed to previous schemes, the node with the "distinguished" hardwired ID ID_l need not be the root of the spanning tree. The final root is determined by the state s set by the adversary at the start of the algorithm—the root is one of the nodes that *believes* in the highest ID.

Chapter 4

A Key Approach to Representing IDs

4.1 The Afek-Matias Probability Distribution

In [AM89], Afek and Matias proposed a probability distribution which can be used to break symmetry in sets of unknown size. Let p be a pair (s, t) of integers, and let pairs be ordered lexicographically. [AM89] proposed a probability distribution on s and t, such that if several (say k) pairs (s_i, t_i) are randomly computed, there is a unique highest pair with probability at least $\bar{\epsilon}$, where $\bar{\epsilon}$ is a constant independent of k. The number s_i is randomly selected according to the probability distribution

$$P(s_i = y) = \frac{1}{2^y}$$

and the number t_i is randomly uniformly selected from the range $[1, 20\ln(4r)]$ where $r = 1/\epsilon$ ($\bar{\epsilon} = 1 - \epsilon$). ϵ is the probability of *error* we are prepared to tolerate for a given collection of randomly chosen values of t_i—with probability $\leq \epsilon$, such a collection will not have a unique maximum). The purpose of t_i is to break symmetry between pairs that have the same s_i, since a small constant number of pairs are expected to have the same highest s_i. For our purposes, we choose $\bar{\epsilon} = \epsilon = 1/2$, so s_i is chosen from the range $[1, 20\ln 8]$. The choice of ϵ affects the

running time of our randomized algorithm by only a constant factor; we have not attempted to compute the optimal value. Our choice of ϵ implies that if k pairs are flipped, there is exactly one highest pair with probability $\geq 1/2$.

Since the protocols and the time complexity analysis do not need to access the individual components of a pair, to ease the notation, we will henceforth assume that a pair (s, t) is uniquely represented as a single integer x. The mapping must preserve the order on (s, t); since the range of t is finite, it is easy to construct such a mapping.

We now formally describe the Afek-Matias probability spaces that we will use. Let Γ_{AM}^k denote the probability space that represents the outcome of k independent pair flips. Let $X_1, X_2, \ldots, X_k$ be independent identically distributed random variables on this space representing the k flips. The distribution of each X is the AM distribution specified earlier; let $P(X = x)$ be denoted by $P_\Gamma(x)$. A sample point p on this space is an outcome of k flips, $(p_1, p_2, \ldots, p_k)$. The set of events on this space is the set 2^O, where O is the set of integer k-tuples. Let $P_{\Gamma^k}(E)$ be the probability of event E. Let $Highest$ be the random variable that returns the highest coin flip:

$$Highest(p_1, p_2, \ldots, p_k) \triangleq \max(p_1, p_2, \ldots, p_k)$$

Also, we define the event UNIQH to be the event that "there exists a unique highest coin flip"; thus

$$\text{UNIQH} \triangleq \{p \mid (\exists i \mid p_i > p_j \forall j \neq i)\}$$

We now state some properties of Γ_{AM}^k. The first property is the main result of [AM89]:

Theorem 4.1 *For any* k, $P_{\Gamma^k}(\text{UNIQH}) \geq 1/2$. ∎

The next two theorems are proved in appendix A:

Theorem 4.2 *For any* k, i, $P_{\Gamma^k}(\text{UNIQH} \mid (Highest > i)) \geq 1/2$. ∎

Theorem 4.3 *For any* k, i, $P_{\Gamma^k}(Highest \neq i) \geq (1 - e^{-1/4}) > 0.22$. ∎

4.2 ID Representation

IDs are represented as tuples of *entries*; each entry is an integer. In the randomized protocol, an entry may represent the result of a number randomly chosen according to the AM scheme (cf. Section 4.1).

We impose a lexicographic order $\prec$ on IDs; this order is a total order. Thus if $X = (x_1, \ldots, x_j)$ and $Y = (y_1, \ldots, y_k)$ are two IDs, then

$$X \prec Y \iff \quad j < k \text{ and } (x_1, \ldots, x_j) = (y_1, \ldots, y_j)$$

$$\text{OR}$$

$$\exists m \leq \min(j, k) \mid (x_1, \ldots, x_{m-1}) = (y_1, \ldots, y_{m-1}) \text{ and } x_m < y_m$$

If the first case holds, i.e. if X is a proper prefix of Y, we define the precedence to hold in the *weak sense*, or $X \stackrel{w}{\prec} Y$. In the second case, X is not a prefix of Y; we define the precedence to hold in the *strong sense*, or $X \stackrel{s}{\prec} Y$. We define the relations $\stackrel{w}{\preceq}$ and $\stackrel{s}{\preceq}$ similarly, but they also include equality (i.e. same IDs).

The *concatenation* of two IDs $X = (a_1, \ldots, a_j)$ and $Y = (b_1, \ldots, b_j)$, written $X : Y$, is defined as the ID $(a_1, \ldots, a_j, b_1, \ldots, b_j)$.

For an ID X, let IDLENGTH(X) denote the number of entries in X, and let $X[i]$ denote the ith entry of X. Let $X[1..i]$ denote the prefix $(X[1], X[2], \ldots, X[i])$.

We now state some basic properties of our ID representation:

Proposition 4.4 *For any IDs A, B, A', B', and C, the following properties hold:*

1. $A \stackrel{w}{\prec} A{:}B.$

2. $(A \stackrel{w}{\preceq} B) \wedge (B \stackrel{w}{\prec} C) \implies (A \stackrel{w}{\prec} C).$

3. $(A \stackrel{s}{\prec} B) \wedge (B \preceq C) \implies (A \stackrel{s}{\prec} C).$

4. $(A \stackrel{s}{\prec} B) \wedge (A \stackrel{w}{\preceq} C) \implies (C \stackrel{s}{\prec} B)$.

5. $(A \stackrel{s}{\prec} B) \implies (A{:}A' \stackrel{s}{\prec} B{:}B')$.

∎

4.3   Motivation behind our ID Representation

As mentioned earlier, nodes *compete* with one another for being the root of the eventual spanning tree. The competition is on the basis of IDs; a higher ID "beats" a lower one; correspondingly, a tree with a high root ID *overruns* a tree with a lower root ID. If two trees with the same root ID detect each other's existence, their root nodes need to break the symmetry so that only one of the two advances in the competition. A highly desirable model to impose on this competition is the *tournament* model, to pick a unique winner starting with n competitors. As the tournament progresses, we have a *shrinking pool* of "candidates" for the eventual winner; once a player leaves the pool, it is out of the running.

Our definition of IDs and the ordering defined on them captures the tournament model. A root can only change its ID by *appending* an *entry* to it. When two roots with equal IDs independently extend their IDs in this manner, one of the new IDs is ordered higher than the other (if they are different). Further, note that the first ID is now higher in the *strong* sense: if the roots perform further (possibly none) extensions, the first root ID will remain higher even after additional extensions (by Proposition 4.4(5)). The second root, with the lower ID, can never compete with the first root after this extension. Hence there exists a shrinking pool of "candidate" roots. The fact that a root "beaten" in this manner cannot compete further for being the eventual root is crucial to the time complexity of our algorithm, since competitions between non-candidate roots do not contribute to the overall time complexity.

If, on the other hand, ID X is higher than ID Y in the *weak* sense, it is still possible for Y, through some sequence of extensions, to eventually be higher than X in the strong sense. Thus a weak-sense relationship between two IDs implies that the roots possessing those IDs are not

yet "differentiated" in the competition; either of them might eventually "beat" the other.

Chapter 5

Specification of the Randomized Algorithm

Section 3 described the basic approach used by our randomized algorithm. This section states the algorithm. The deterministic version is very similar to the randomized one; we briefly describe the deterministic version in Section 8.

The network can be modeled as a probabilistic automaton RSST (for "**R**andomized **S**elf-stabilizing **S**panning **T**ree") whose state s contains a global time component $s.now$, a set of deadlines $\{s.deadline(a)\}$ (cf. Section 2.3.1), and the states of the network nodes. The state $s_{[u]}$ of each node u consists of a set of shared variables ID_u, $distance_u$, $parent_u$, $color_u$, $mode_u$, $other\text{-}trees_u$, and, for each neighbor v of u, $nbr\text{-}color_{uv}$. In addition, the state of each node u contains a set of local variables ID_{uv}, $distance_{uv}$, $parent_{uv}$, $color_{uv}$, $mode_{uv}$, $other\text{-}trees_{uv}$ and $self\text{-}color_{uv}$ for each neighbor v of u; these are local copies of the corresponding variables at v (with the exception of $self\text{-}color_{uv}$, which is a local copy of $color_{vu}$) which node u maintains and periodically updates by reading v's shared variables. These variables can be partitioned into two categories: those associated with *tree overrunning*—*ID, distance, parent*; and those associated with *recoloring* or the process of *detecting "competing" trees*—*color, mode, other-trees, self-color,* and *nbr-color* (cf. Section 3). The state variables and their types are listed in

Shared variables:

Variables for tree overrunning:
$ID_u \in$ ID-tuples (cf. Section 4), current ID
$distance_u \in \{0, 1, 2...\}$, estimate of current distance from root
$parent_u \in \{$nil$\} \cup Nbrs(u)$, pointer to *parent*

Variables for tree recoloring:
$color_u \in \{0, 1, 2, 3\}$, current color
$mode_u \in \{$broadcast, echo$\}$, recoloring phase
$other\text{-}trees_u \in \{$true, false$\}$, existence of other trees with same ID

$\forall v \in Nbrs(u)$, $nbr\text{-}color_{uv} \in \{1, 2, 3,$ undefined$\}$, last "real" color of nbr v

Local variables:

$\forall v \in Nbrs(u)$,

/* local copy of corresponding shared variables at neighbor v */
ID_{uv}, $parent_{uv}$, $distance_{uv}$, $color_{uv}$, $mode_{uv}$, $other\text{-}trees_{uv}$, $self\text{-}color_{uv}$

(Note that $color_{uv}$, $self\text{-}color_{uv} \in \{0, 1, 2, 3,$ undefined$\}$)

Figure 5-1: Set $s_{[u]}$ — State components of node u

Figure 5-1.

Nodes maintain *ID*s; these IDs are *not* "hard-wired" (since we are considering anonymous networks here), and are susceptible to *change*. The $parent_u$ variable at u points to a neighboring node (or "nil"); the set of $parent_u$ variables at all nodes $u \in V$ define a subset E_{parent} of the set of edges E. We attempt to make the *parent subgraph* $G_{parent} = (V, E_{parent})$ represent a *forest*; thus we attempt to make each node u belong to a *tree* T_u. The $distance_u$ variable is an estimate of the distance from u to the root of its tree T_u (if such a tree exists).

The *priority* of a node u is defined to be the tuple *$(ID_u, distance_u)$*. We define a total order $\gg$ on priorities:

```
/* copy neighbor variables into local memory */
∀v ∈ Nbrs(u), COPY_uv

/* become child of neighbor with maximum priority, or become root */
MAXIMIZE-PRIORITY_u

/* if local neighborhood "looks" stable, participate in recoloring etc.*/
DETECT-TREES_u

/* if root's tree has acknowledged color, choose new color */
NEXT-COLOR_u

/* if root has detected other trees with same ID, extend ID */
EXTEND-ID_u
```

Figure 5-2: Actions of node u

Definition 5.1 (Order $\gg$ on priorities) $(ID_u, distance_u) \gg (ID_v, distance_v)$ iff either $ID_u > ID_v$, or their IDs are equal and $distance_u < distance_v$. The analogous relation $\underline{\gg}$ includes equality. ∎

The protocol at each node u is implemented through the atomic *actions* specified in Figure 5-2. Note that each action is always enabled; actions need not be performed in any particular order. At each state of an execution $\alpha = s_0 a_1 s_1 a_2 s_2 ...$, the adversary chooses the next processor u to perform an action, as well as the particular action of u that is performed.

The action COPY_{uv} (Fig. 5-3) reads the values of neighbor v's shared variables and copies it into the corresponding local "opinions" at node u. Besides, it performs tasks related to the coloring algorithm. The action $\text{MAXIMIZE-PRIORITY}_u$ (Fig. 5-4) makes u participate in the important task of tree overrunning; it sets the *ID, distance* and *parent* variables. (It makes node u maximize its *priority* by attaching to neighboring nodes, if possible.) The action DETECT-TREES$_u$ (Fig. 5-5) makes u participate in recoloring its tree to detect "competing" trees with

39

the same ID. If u is a root whose tree has acknowledged being colored with a certain color, the action NEXT-COLOR$_u$ (Fig. 5-6) makes u choose the next color to color its tree with. Finally, if u is a root node and the recoloring process has informed it of a "competitor" tree, the action EXTEND-ID$_u$ (Fig. 5-7) causes u to *extend* its ID randomly to break symmetry.

Definition 5.2 (RSST) The probabilistic automaton RSST is defined as follows:

1. The set *states*(RSST) consists of all states s such that

   - The values of all variables in $s_{[u]}$ belong to their corresponding types (listed in Figure 5-1),

   - $s.now \geq 0$, and

   - for each $a \in acts($RSST$)$, $s.deadline(a) \geq 1$.

2. $start($RSST$) = \{s \mid s.now = 0 \wedge \ \forall \ a \in acts($RSST$), s.deadline(a) = 1\}$.

3. $acts($RSST$) = \nu$ (time passage), and for all u and all $v \in Nbrs(u)$, $\{$COPY$_{uv}$, MAXIMIZE-PRIORITY$_u$, DETECT-TREES$_u$, NEXT-COLOR$_u$, EXTEND-ID$_u\}$.

4. *steps*(RSST) is specified by the code for the individual actions in *acts*(RSST), listed in Figures $5\text{-}3 - 5\text{-}7$.   ∎

Henceforth, the code is organized, for convenience, into *statements* labeled [A], [B], [C], etc.

Statement [**A**] in action COPY$_{uv}$ (Figure 5-3) invoked by node u performs the task of reading the shared variables of the neighbor v and copying them into local memory. For example, the value of ID_u at node u is the value of u's current ID, and ID$_{uv}$ is intended to hold the latest "opinion" of the ID of neighbor v. Statements [**B**], [**C**] and [**D**] perform tasks required for the tree detection algorithm; they are described in Section 5.1.

We want trees with high root IDs to "overrun" trees with lower root IDs. To this end, each node u tries to "optimize" its ID: if it notices a neighbor with an ID higher than itself, it

```
COPY_uv

/* make local copies of neighbor variables */                                          [A]


        ID_uv ← ID_v
        distance_uv ← distance_v
        parent_uv ← parent_v
        color_uv ← color_v
        mode_uv ← mode_v
        other-trees_uv ← other-trees_v
        self-color_uv ← color_vu



/* perform coloring tasks if necessary */
if (ID_v = ID_u and |(distance_v − distance_u)| ≤ 1)                                    [B]
then

        /* record color of neighbor if necessary */                                    [C]
        if (color_u ≠ 0) and (color_v ≠ 0)
        then
                nbr-color_uv ← color_v
                if color_v ≠ color_u then other-trees← true

        /* copy parent's color if necessary */                                         [D]
        if (parent_u = v) and (color_u ≠ color_v)
        then Reset-Color_u(color_v)
```

Figure 5-3: Action COPY$_{uv}$

MAXIMIZE-PRIORITY$_u$

/* let l be the "largest" of all neighbors that have max priority */
Let $l \leftarrow \max \{x \mid (ID_{ux}, distance_{ux}) = \max'_{v \in Nbrs(u)}(ID_{uv}, distance_{uv}) \}$ **[E]**
(where max$'$ is maximum over the relation $\gg$, cf. Definition 5.1)

/* force root to extend first, if about to be overrun by a suffix ID */
if $(parent_u = \text{nil})$ **and** $ID_u \overset{w}{\prec} ID_{ul}$ **then**   **[F]**
    while $ID_u \overset{w}{\prec} ID_{ul}$
        Append-Entry$_u()$

/* if u can improve its priority, by becoming child of another */
/* neighbor, do so, otherwise become root */

if $(ID_{ul}, distance_{ul}) \gg (ID_u, distance_u)$ /* see def. of $\gg$ */
then   **[G]**
    $ID_u \qquad\quad \leftarrow ID_{ul}$
    $distance_u \quad \leftarrow distance_{ul} + 1$
    $parent_u \qquad \leftarrow l$

else /* no neighbor has a larger priority; become root */   **[H]**
    $distance_u \quad \leftarrow 0$
    $parent_u \qquad \leftarrow \text{nil}$

Figure 5-4: Action MAXIMIZE-PRIORITY$_u$

attaches to the neighbor with the highest ID, and changes its ID to the observed ID. Further, once it has optimized its ID, it also tries to optimize its *distance*: it prefers to attach to the node with the smallest *distance*. The purpose of the *distance* counters is to "shrink" long branches in trees so that no branch can exceed diameter length. Hence in [**E**] of MAXIMIZE-PRIORITY, we make u determine the neighbor l with the highest priority. Many neighbors may all have the same highest priority; we break ties by choosing the highest-ordered neighbor (each processor is assumed to maintain a total order on its neighbors, so that such ties can be resolved in a consistent manner).

The purpose of statement [**F**] is rather technical; it is not required for correctness but plays an important role in maintaining an overall $O(\delta)$ time complexity for our algorithm. (Note that δ is the network diameter.) As will be explained in the time complexity analysis of Section 6, statement [**F**] limits the power of an adversary to alter the probability distribution of existing root IDs.

Statement [**G**] determines whether node u can *increase* its priority by attaching to the "highest" neighbor l determined by [**E**]. If the priority cannot decrease, it then makes the neighbor l its *parent*, assumes its ID, and assumes its distance incremented by one.

However, if node u *can only decrease* its priority by attaching to the neighbor l, [**H**] makes it *become a root*, keeping its ID unchanged and resetting its *distance* to zero. This is the mechanism of handling the *ghost root* problem described earlier—if node u notices that it was a nonroot node with a ID I_g that is *not* possessed by any of its neighbors and is higher than all its neighbors IDs, it was erroneously "believing" in the existence of a root node with ID I_g. In this situation, node u simply becomes a root with ID I_g by setting its *distance* to zero, thus obviating the need to "correct" erroneous belief in that root elsewhere in the network. Hence statement [**H**] plays an important role in self-stabilization.

43

5.1 The Tree Detection Algorithm

The Tree Detection Algorithm has the following purpose: if two or more neighboring trees have the same root ID, we want their roots to detect this condition, so that they can then extend their IDs to break symmetry and advance in the competition. The complexity in the code arises from having to contend with faults, asynchrony, and the fact that we regard the scheduler as an *adversary* capable of altering the schedule to thwart our intentions. The Tree Detection Algorithm is implemented through statements [**B**], [**C**] and [**D**] in action COPY, and through actions DETECT-TREES, NEXT-COLOR and EXTEND-ID.

Statements [**B**] and [**I**] test for a "stability condition"; the rest of the tree detection code in actions COPY and DETECT-TREES is only executed if the neighborhood of node u appears to "believe in" only one ID. If this is not the case, tree overrunning is still in progress in the neighborhood of u, and so tree detection can not be performed.

Let node u belong to a tree T defined on the parent subgraph. (As will be shown in the proof, the action MAXIMIZE-PRIORITY guarantees that u eventually belongs to some tree.) Let the root of T be node r_u. The tree detection algorithm *colors* the tree T with an alternating sequence of colors $\{\ c_1, 0, c_2, 0, c_3, 0, ...\ \}$, where $c_i \neq 0$ for all i. The *color* variable of a node represents its current color.

Let the *color* of the root r_u at some instant be c. Nodes in the tree propagate color c to their children, so that eventually all nodes in tree T will set their *color* to c. When the entire tree is colored with c, nodes *acknowledge* this fact to the root. This propagation and acknowledgement is done through a standard "broadcast-echo" mechanism: the *mode* field of a node is set to either *broadcast* or *echo*, depending on which phase of the recoloring is in progress at that node.

When a node notices that its own color is different from that of its parent (in statement [**D**]), it calls the subroutine Reset-Color$_u$ (Fig. 5-8), which "resets" its coloring variables, and causes it to broadcast its parent's color (by setting its *mode* to broadcast and copying its parent's color). In this manner, when a root r chooses a new color, its descendants successively

44

DETECT-TREES$_u$

if $\forall v \in Nbrs(u)$, $(ID_{uv} = ID_u$ **and** $|(distance_{uv} - distance_u)| \leq 1)$ **[I]**

then

    /* check for echo */
    if **[J]**
        { $(mode_u = $ broadcast$)$

            /* and if all children echo v's color */
            and $(\forall v \in$ Children$_u$, $mode_{uv} = $ echo and $color_{uv} = color_u$ $)$

            /* and if "mirror technique" is applicable : see text. If node u has */
            /* some color $(\neq 0)$, it should have observed neighbors' colors, and */
            /* neighbors should have observed u's color, detected by $self\text{-}color_{uv}$ */
            and $(color_u \neq 0 \implies \forall v \in Nbrs(u)$,
                $nbr\text{-}color_{uv} \neq$ undefined and $self\text{-}color_{uv} = color_u)$
        }
    then **[K]**
        $mode_u \leftarrow$ echo
        if $(\exists v \in$ Children$_u$ | $other\text{-}trees_{uv} = $ true$)$ then $other\text{-}trees_u \leftarrow$ true

Figure 5-5: Action DETECT-TREES$_u$

NEXT-COLOR$_u$

    /* If root, choose new color if necessary */
    if $(parent_u = $ nil **and** $mode_u = $ echo **and** $other\text{-}trees_u = $ false$)$
    then
        Reset-Color$_u$(New-Color$_u$())

Figure 5-6: Action NEXT-COLOR$_u$

```
EXTEND-ID_u

    if (parent_u = nil and mode_u = echo and other-trees_u = true)
    then
        Append-Entry_u()
        Reset-Color_u(New-Color_u())
```

Figure 5-7: Action EXTEND-ID$_u$

copy that color, and a "broadcast wave" propagates throughout T.

In a simple echoing scheme that does not need to take into account an adversarial scheduler, each node u sets its direction to "echo" when all its children are echoing the same color (i.e. all children have the same *color* c as node u and have their *mode* set to *echo*). This is also part of our condition for echoing, which is tested in [**J**]. In this manner, an "echo wave" travels upwards from the leaves to the root.

When the root r_u notices that all its children are echoing its color c, it concludes that its entire tree is colored with c, and then changes its color (through action NEXT-COLOR, in Fig. 5-6). Its new color is a function of the previous color c: it alternates between 0 and a color randomly chosen from $\{1, 2, 3\}$. The rationale for the coloring sequence was described in Section 3.

When a node is broadcasting some color (i.e., $mode_u$ = broadcast), it *checks for the existence of competing trees with the same ID*. This check is performed in [**C**]. In the scheme for a non-adversarial scheduler, if a node observes that some neighbor is colored with a color different from its own (provided neither color is 0), it can correctly conclude that that neighbor belongs to a tree different from itself. If node u detects such a competing tree, it sets its *other-trees* to *true*; the echoing mechanism conveys this information to the root of the tree (through statement [**K**]). If a root is thus informed of the existence of a competing tree (i.e. another tree with the same root ID), it attempts to break symmetry by extending its ID (action

46

Append-Entry$_u$()

    $ID_u \leftarrow ID_u$:x

        where x is an entry chosen by the Afek-Matias [AM89] scheme

New-Color$_u$()

    if $color_u = 0$

    then return color randomly chosen from $\{1, 2, 3\}$

    else return 0

Reset-Color$_u$(*color*)   /* reset local recoloring-related variables */

    $color_u$ $\leftarrow color$

    $mode_u$ $\leftarrow$ broadcast

    $other\text{-}trees_u \leftarrow$ false

    $\forall v \in Nbrs(u)$,

        $nbr\text{-}color_{uv} \leftarrow$ undefined

        $self\text{-}color_{uv} \leftarrow$ undefined

    $\forall v \in$ Children$_u$, $color_{uv} \leftarrow$ undefined

Children$_u$: $\{\ v \mid parent_{uv} = u\ \}$

Figure 5-8: Macros

EXTEND-ID, Fig. 5-7). After extending its ID the root participates in the overrunning and recoloring processes all over again.

However, this scheme of detecting duplicate IDs (i.e. u is colored with a non-zero color different from that of some neighbor v implies that v is in a different tree) is not sufficient if the scheduler is *adversarial.* Consider the recoloring process operating on two neighboring trees T and $\overline{T}$ having the same root ID, containing two neighboring nodes u and v respectively. We want our tree detection process to eventually let at least one of the trees detect this situation. However, the schedule could be manipulated by the adversary such that the two trees are never *both* colored with a non-zero color; the adversary could schedule steps such that always exactly one of the trees is colored 0 and the other is colored with a non-zero color. In such a schedule, the trees can continue the recoloring process indefinitely without ever detecting each other.

An idea proposed in [AKY90] modifies the scheme so that it can accomodate an adversarial scheduler. The idea is that when a node u is colored with a non-zero color, it *waits for each neighboring node to be colored with a non-zero color*, and records this color individually for each neighbor v as soon as available, in the variable *nbr-color$_{uv}$* (in [C]). Correspondingly, it waits till it observes that each neighbor v has observed its own color, by examining the variable *self-color$_{uv}$*, which it copied from its neighbor. The test for this mirror-like scheme is part of the condition [J] for echoing. Section 7 shows that this scheme succeeds for the adversarial scenario described earlier.

Chapter 6

Correctness and Complexity Proof for the Randomized Algorithm: Part 1

The probabilistic automaton RSST implementing our randomized protocol was defined in Definition 5.2. We prove that RSST constructs a spanning tree within expected $O(\delta)$ time, where δ is the network diameter. In this section we give some basic definitions and an overview of the proof.

6.1   Spanning Trees

We first define the states of RSST that define a spanning tree.

Definition 6.1 For any $s \in states(\mathsf{RSST})$,

- $\xi(s)$ is the multiset of the node ids in s, i.e.

$$\xi(s) \;\triangleq\; s.\{ID_{v_1}, ID_{v_2}, ID_{v_3}, \ldots, ID_{v_n}\}$$

- Node u is a *root* in state s if $s.parent_u = $ nil.

- The set $\rho(s)$ is the set of root nodes in s, i.e.

$$\rho(s) \triangleq \{u \in V \mid s.parent_u = \text{nil}\}$$

- Node u is an *ancestor* of node v $(u \neq v)$ in s if there exists a sequence of nodes $\{u, u_1, u_2, \ldots, u_j, v\}$ such that $parent_{u_1} = u, parent_{u_2} = u_1, \ldots, parent_{u_j} = u_{j-1}, parent_v = u_j$.

- State s *contains a cycle* if there exists a node that is an ancestor of itself.

- State s *defines a forest* if it does not contain a cycle.

- State s *defines a spanning tree* if it defines a forest and $|\rho(s)| = 1$. ■

Let the set

$$\mathcal{S} \triangleq start(\mathsf{RSST})$$

denote the set of start states of RSST. The set $\mathcal{ST}$ is defined as the set of states defining a spanning tree. Thus,

$$\mathcal{ST} \triangleq \{s \in states(\mathsf{RSST}) \mid s \text{ defines a spanning tree}\}$$

6.2 Overview of the Proof

In this section we give an outline of the proof. We need to prove that departing from a state of $\mathcal{S}$, the expected time to reach a state of $\mathcal{ST}$ is $O(\delta)$.

Our proof is divided into several phases, each one of which proves a property of making a partial time bounded progress toward a "success state", i.e., a state of $\mathcal{ST}$. The state sets

associated with the different phases are $\mathcal{S}$, $\mathcal{F}'$, $\mathcal{F}$, $\mathcal{C}^=$, $\mathcal{C}^1$, $\mathcal{G}$, and $\mathcal{ST}$. Here,

$$\mathcal{F}' \triangleq \{s \mid s \text{ defines a forest}\}$$

is the set of forest-defining states, and

$$\mathcal{F} \triangleq \left\{ s \quad \middle| \quad \forall u, v, \begin{bmatrix} 1. & v = s.parent_u \implies s.(ID_u, distance_u) \ll s.(ID_{uv}, distance_{uv}) \\ 2. & v \in Nbrs(u) \implies s.(ID_{uv}, distance_{uv}) \lessapprox s.(ID_v, distance_v) \end{bmatrix} \right\}$$

is a subset of the set of closed forest-defining states (this property will be shown in Section 6.3). Thus, once a state of $\mathcal{F}$ is reached, the global state always defines a forest.

To motivate the definitions of $\mathcal{C}^=$, $\mathcal{C}^1$, and $\mathcal{G}$, we introduce the set

$$\Phi(s) \triangleq \{u \in \rho(s) \mid \neg(\exists v \in \rho(s) \mid ID_u \overset{s}{\prec} ID_v)\}$$

of "candidate" roots in state s. This set plays a crucial role in maintaining progress of our algorithm. As mentioned in the description of the algorithm, root nodes compete for being the root of the eventual spanning tree. We show that the root of the eventual spanning tree must *always* be present in Φ after time 2, and moreover, that Φ can only *shrink* with time (and thus $|\Phi|$ can never increase). These properties imply that if a state is in the set of "good" states

$$\mathcal{G} \triangleq \{s \in \mathcal{F} \mid (\, |\Phi(s)| \, = 1)\}$$

then the root of the final spanning tree is uniquely determined. Let s be a state in $\mathcal{F}$ such that $s \notin \mathcal{G}$. Since $|\Phi(s)| > 1$, for achieving progress we need to show that starting from a state in $\mathcal{F}$, $|\Phi|$ is reduced to 1 (i.e., a state in $\mathcal{G}$ is reached) in expected $O(\delta)$ time. We do so using the intermediate state sets $\mathcal{C}^=$ and $\mathcal{C}^1$. $\mathcal{C}^=$ is defined as the set of states

$$\mathcal{C}^= \triangleq \{s \in \mathcal{F} \mid \forall u, v \in \Phi(s), ID_u = ID_v\}$$

51

$$\begin{array}{ll} \mathcal{S} \xrightarrow{3} \mathcal{F} & (\text{Proposition } 6.11) \\[4pt] \mathcal{F} \longrightarrow \mathcal{F}_{\boxdot} & (\text{Proposition } 6.12) \\[4pt] \mathcal{F} \xrightarrow{2\delta} \mathcal{C}^= \cup \mathcal{C}^1 & (\text{Proposition } 6.24) \\[4pt] \mathcal{C}^= \xrightarrow[2/9]{77\delta+36} \mathcal{C}^1 & (\text{Proposition } 7.80) \\[4pt] \mathcal{C}^1 \xrightarrow[0.11]{2\delta} \mathcal{G} & (\text{Proposition } 6.49) \\[4pt] \mathcal{G} \xrightarrow{2\delta} \mathcal{ST} & (\text{Proposition } 6.23) \end{array}$$

Figure 6-1: Proof Phases

in which the IDs of all candidate roots are equal. To define $\mathcal{C}^1$, we first define subsets $\Phi_i(s)$ of Φ as follows:

$$\Phi_i(s) \triangleq \{u \in \Phi(s) \mid \text{IDLENGTH}(ID_u) = i\}$$

$\Phi_i(s)$ is defined as that subset of $\Phi(s)$ whose elements have IDs of a particular length i. (The set $\Phi_{>i}(s)$ contains elements having IDs of length greater than i; $\Phi_{<i}(s)$ is defined similarly.) We define the special subset

$$\Phi_{l_{max}}(s) \triangleq \{\max_i(\Phi_i \mid \Phi_i \neq \phi)\}$$

as that subset of Φ whose elements have IDs of maximal length. Finally, we are in a position to define $\mathcal{C}^1$ as

$$\mathcal{C}^1 \triangleq \{s \in \mathcal{F} \mid |\Phi_{l_{max}}(s)| = 1\}$$

i.e., $\mathcal{C}^1$ is the set of states in which there is just one element in Φ whose ID is of maximal length.

Having defined the relevant state sets, we now formally describe the phases of our proof; they are summarized in Figure 6-1.

The first statement states that starting from a start state, a forest-defining state is reached within time 3; the second statement states that once a forest-defining state is reached, the

state always defines a forest. The last statement states that once a "good" state is reached, within time 2δ the state defines a spanning tree.

By combining the statements above using Theorem 2.11 and Corollary 2.12, we obtain

$$\mathcal{F} \xrightarrow[0.025]{81\delta+36} \mathcal{G}$$

and consequently

$$\mathcal{S} \xrightarrow[0.025]{83\delta+39} \mathcal{ST}$$

Using the results of the proof summary above, we can derive an upper bound of $O(diameter)$ on the expected time required to reach a state of $\mathcal{ST}$ starting from a state of $\mathcal{S}$.

Theorem 6.2 *Under any fair adversary, starting from any start state, the automaton* RSST *that implements our randomized self-stabilizing spanning tree algorithm reaches a state defining a spanning tree within expected $O(\delta)$ time.*

Proof. Departing from a state in $\mathcal{F}$, RSST reaches a state in $\mathcal{G}$ in time (at most) $81\delta+36$ with probability at least 0.025. Consider an execution of RSST starting from a state s in $\mathcal{F}$, and consider successive *epochs* of duration $81\delta+36$. In the first epoch, the probability of attaining membership in $\mathcal{G}$ ("success in the first epoch") is at least 0.025. Since $\mathcal{F}$ is closed, the probability of success in every such epoch is at least 0.025. Hence, the expected number of epochs needed to attain success has an upper bound of $\lceil 1/0.025 \rceil$, or 40. Hence, starting from a state in $\mathcal{F}$, the expected time taken to reach a state in $\mathcal{G}$ has an upper bound of $40 * (81\delta + 36)$, which is $O(\delta)$. Since $\mathcal{S} \xrightarrow{3} \mathcal{F}$ and $\mathcal{G} \xrightarrow{2\delta} \mathcal{ST}$, the expected time to reach a state in $\mathcal{ST}$ starting from a state in $\mathcal{S}$ is $O(\delta)$. ∎

We now proceed with the details of the proof, i.e. the proofs of the probabilistic statements given above. Let $\mathcal{A} \in$ *Fairadvs* be a fair adversary for RSST. Let $z \in \mathcal{S}$ be an arbitrary starting state. Let $\hat{H}$ denote the execution automaton $H(\mathsf{RSST}, \mathcal{A}, z)$. Let α' denote an execution of $\hat{H}$, and let α be the corresponding execution $\alpha' \downarrow$ of RSST.

In Section 6.3, we prove the statements $\mathcal{S} \xrightarrow{3} \mathcal{F}$, $\mathcal{F} \longrightarrow \mathcal{F}_{\square}$, $\mathcal{F} \xrightarrow{2\delta} \mathcal{C}^= \cup \mathcal{C}^1$, and $\mathcal{G} \xrightarrow{2\delta} \mathcal{ST}$. The statement $\mathcal{C}^= \xrightarrow[2/9]{77\delta+36} \mathcal{C}^1$ is proved in Section 7, and the statement $\mathcal{C}^1 \xrightarrow[0.11]{2\delta} \mathcal{G}$ is proved in Section 6.4.

6.3 Stabilization of Forest Structure, Candidate Root Properties

In this section we prove the statements $\mathcal{S} \xrightarrow{3} \mathcal{F}$, $\mathcal{F} \longrightarrow \mathcal{F}_{\square}$, $\mathcal{F} \xrightarrow{2\delta} \mathcal{C}^= \cup \mathcal{C}^1$, and $\mathcal{G} \xrightarrow{2\delta} \mathcal{ST}$.

6.3.1 Forest Structure - Establishment and Preservation

Each node v maintains an "opinion" of the values of the shared variables of its neighbors in its own local variables. Claim 6.3 states that after time 1, this "opinion" must have actually been read from the neighbors, i.e. it is no longer arbitrarily set in the start state.

Claim 6.3 *For any s such that $s.now > 1$, $s.VAR_{uv} = s'.VAR_v$ for some s' preceding s in α, where VAR is one of $\{ID, distance, parent\}$.*

Proof. Within time 1, node u will have performed COPY-NBRS$_{uv}$ for all neighbors v, and hence will have read the local variables of all its neighbors at least once. ∎

Claim 6.4 and Lemma 6.5 show that the *priority* (defined as the tuple (*ID*,*distance*)) of a node cannot decrease (in terms of the order $\ll$ defined on priorities; cf. Definition 5.1); if it changes, it can only increase.

Claim 6.4 *For any step $(s, \text{EXTEND-ID}_u, (\Omega, \Sigma, P))$ of RSST, for any state $s' \in \Omega$, $s.ID_u \overset{w}{\preceq} s'.ID_u$.*

Proof. Follows directly from Proposition 4.4(1). ∎

Lemma 6.5 *For any step* $(s, a, (\Omega, \Sigma, P))$ *of* RSST, *for any* $s' \in \Omega$,
$s.(ID_u, distance_u) \lll s'.(ID_u, distance_u)$.

Proof. The only actions which change $(ID_u, distance_u)$ are MAXIMIZE-PRIORITY$_u$ and EXTEND-ID$_u$. If MAXIMIZE-PRIORITY$_u$ is executed, only statements [**F**], [**G**] and [**H**] are capable of changing $(ID_u, distance_u)$. Let the "intermediate" value of ID_u after executing statement [**F**] be I; then $s.ID_u \preceq I$. If [**G**] is executed, the value of $(ID_u, distance_u)$ cannot decrease, because of the direction of the precedence test. [**H**] leaves ID_u intact and sets $distance_u$ to 0; thus $s.ID_u \preceq I = s'.ID_u$ and $s.distance_u \geq s'.distance_u$, and so the priority $(ID_u, distance_u)$ cannot decrease. By Claim 6.4, EXTEND-ID$_u$ increases ID_u, and therefore increases the priority $(ID_u, distance_u)$. ∎

Corollary 6.6 *For all* s *and* s' *such that* s *precedes* s' *in* α,
$s.(ID_u, distance_u) \lll s'.(ID_u, distance_u)$. ∎

Since priorities do not decrease, then, by Claim 6.3, priorities as observed by neighbors do not decrease:

Corollary 6.7 *For any node* u, *any* v, *the value of* $(ID_{uv}, distance_{uv})$ *cannot decrease after time 1.* ∎

We now establish that in any execution, any state after time 2 belongs to the set $\mathcal{F}$, and thus defines a forest.

Lemma 6.8 *For all* s *such that* $s.now > 2$, *each node* u *obeys the* priority invariant:
$(parent_u = v) \implies (ID_u, distance_u) \ll (ID_{uv}, distance_{uv})$.

Proof. Consider any node u which is a child of node v in some state s such that $s.now > 2$; thus $s.parent_u = v$. Node u last executed the statement $(parent_u \leftarrow v)$ in [**G**] at some step $(s_1, \text{MAXIMIZE-PRIORITY}_u, s_2)$, where $(s.now - 1) \leq s_2.now \leq s.now$. By [**G**], we have $s_2.(ID_u, distance_u) = s_1.(ID_{uv}, distance_{uv} + 1)$. Since s_1 precedes s in α and $s_1.now > 1$, by Corollary 6.7, $s_1.(ID_{uv}, distance_{uv}) \lll s.(ID_{uv}, distance_{uv})$. Hence, we have:

$$
\begin{aligned}
s.(ID_u, distance_u) \quad &= \quad s_2.(ID_u, distance_u) \\
&= \quad s_1.(ID_{uv}, (distance_{uv} + 1)) \\
&\ll \quad s_1.(ID_{uv}, distance_{uv}) \\
&\lll \quad s.(ID_{uv}, distance_{uv})
\end{aligned}
$$

Hence $s.(ID_u, distance_u) \ll s.(ID_{uv}, distance_{uv})$. ∎

Corollary 6.9 *For all s such that $s.now > 1$, for any node u and any $v \in Nbrs(u)$,*
$s.(ID_{uv}, distance_{uv}) \lll s.(ID_v, distance_v)$.

Proof. Let the last COPY_{uv} step executed by u be $(s_1, \text{COPY}_{uv}, s_2)$. Then $s.(ID_{uv}, distance_{uv}) = s_2.(ID_{uv}, distance_{uv}) = s_1.(ID_v, distance_v)$. Since s_1 precedes s in α, by Corollary 6.6, $s_1.(ID_v, distance_v) \lll s.(ID_v, distance_v)$. Hence $s.(ID_{uv}, distance_{uv}) \lll s.(ID_v, distance_v)$. ∎

Corollary 6.10 $\mathcal{F} \subseteq \mathcal{F}'$.

Proof. Let $s \in \mathcal{F}$. By the definition of $\mathcal{F}$, for any u, v such that $v = parent_u$, $s.(ID_u, distance_u) \ll s.(ID_v, distance_v)$. Since each node must have a strictly lower priority than its parent, s cannot contain a cycle. ∎

Proposition 6.11 $\mathcal{S} \xrightarrow{3} \mathcal{F}$.

Proof. Immediate from Lemma 6.8 and Corollary 6.9. ∎

Proposition 6.12 $\mathcal{F} \longrightarrow \mathcal{F}_{\boxdot}$.

Proof. Let $(s, a, (\Omega, \Sigma, P))$ be a step of RSST. Let $s \in \mathcal{F}$, and let $s' \in \Omega$. We need to show that $s' \in \mathcal{F}$. Recall the definition of $\mathcal{F}$:

$$
\mathcal{F} \triangleq \left\{ s \;\middle|\; \forall u, v, \begin{bmatrix} 1. & v = s.parent_u \Longrightarrow s.(ID_u, distance_u) \ll s.(ID_{uv}, distance_{uv}) \\ 2. & v \in Nbrs(u) \Longrightarrow s.(ID_{uv}, distance_{uv}) \lll s.(ID_v, distance_v) \end{bmatrix} \right\}
$$

56

The only variables that determine membership in $\mathcal{F}$ are *parent*, *ID*, and *distance* (both local and shared copies). Thus the only actions that can change membership in $\mathcal{F}$ are COPY, MAXIMIZE-PRIORITY and EXTEND-ID.

Case 1 $a = \mathrm{COPY}_{uv}$.

The only relevant effect is that $s'.(ID_{uv}, distance_{uv}) = s'.(ID_v, distance_v)$; thus predicate 2 of the definition of $\mathcal{F}$ holds for u. If $v = parent_u$, then

$$
\begin{aligned}
s'.(ID_u, distance_u) \quad &= \quad s.(ID_u, distance_u) \\
&\ll \quad s.(ID_{uv}, distance_{uv}) \quad \text{(since } s \in \mathcal{F}) \\
&\lleq \quad s'.(ID_{uv}, distance_{uv}) \quad \text{(by Corollary 6.7)}
\end{aligned}
$$

Hence $s'.(ID_u, distance_u) \ll s'.(ID_{uv}, distance_{uv})$, and predicate 1 holds. Since no other node predicates are affected, $s' \in \mathcal{F}$.

Case 2 $a = \mathrm{MAXIMIZE\text{-}PRIORITY}_u$.

The only variables set are ID_u, $distance_u$, and $parent_u$, so we only need to check that in state s', u satisfies predicate 1, and that all neighbors of u satisfy predicate 2. Either statement $[\mathbf{G}]$ or $[\mathbf{H}]$ of MAXIMIZE-PRIORITY$_u$ must be executed. If $[\mathbf{G}]$ is executed, $s'.(ID_u, distance_u) = s'.(ID_{ul}, (distance_{ul} + 1)) \ll s'.(ID_{ul}, distance_{ul})$, where $l = s'.parent_u$. Hence u satisfies predicate 1. If $[\mathbf{H}]$ is executed, u trivially satisfies predicate 1 in s'. For any $v \in Nbrs(u)$,

$$
\begin{aligned}
s'.(ID_{vu}, distance_{vu}) \quad &= \quad s.(ID_{vu}, distance_{vu}) \\
&\lleq \quad s.(ID_u, distance_u) \quad \text{(since } s \in \mathcal{F}) \\
&\lleq \quad s'.(ID_u, distance_u) \quad \text{(by Corollary 6.6)}
\end{aligned}
$$

Thus any neighbor v satisfies predicate 2, and hence $s' \in \mathcal{F}$.

Case 3 $a = \mathrm{EXTEND\text{-}ID}_u$.

If ID_u is extended, $u \in \rho(s')$, so u trivially satisfies predicate 1 in state s', and since $s \in \mathcal{F}$, u satisfies predicate 2 in s'. By an argument identical to that for Case 2, all neighbors v of u also satisfy the predicates, and so $s' \in \mathcal{F}$. ∎

Henceforth in the proof, for all states mentioned we will assume that $s \in \mathcal{F}$. Thus each state under discussion defines a forest.

We now show that the set of root nodes $\rho(s)$ can only diminish with time—a root may become a nonroot, but not vice-versa.

Lemma 6.13 $\rho(s') \subseteq \rho(s)$ *for all* s, s' *such that* s' *follows* s *in* α.

Proof. Suppose not, i.e. suppose $\exists u$ such that $u \in \rho(s')$ but $u \notin \rho(s)$. Then $s'.parent_u = $ nil and $s.parent_u \neq $ nil. Hence there must exist a step $(s_3, \text{MAXIMIZE-PRIORITY}_u, s_4)$ in α, such that $s_3.parent_u \neq $ nil, $s_4.parent_u = $ nil, and [**H**] was executed in $\text{MAXIMIZE-PRIORITY}_u$. Let $s_3.parent_u = v$. From the test that causes [**H**] to be executed, $s_3.(ID_{uv}, distance_{uv}) \lll s_3.(ID_u, distance_u)$. (Note that since $s_3.parent_u \neq $ nil, [**F**] was not executed in this step.)

But since $s_3.parent_u = v$, there must exist a preceding step $(s_1, \text{MAXIMIZE-PRIORITY}_u, s_2)$ in which $(s_2.(ID_u, distance_u) = s_3.(ID_u, distance_u))$ and $parent_u$ was set to v. Since [**G**] was executed in this step, $s_2.(ID_{uv}, distance_{uv}) \ggg s_2.(ID_u, distance_u)$. By Corollary 6.7, $s_3.(ID_{uv}, distance_{uv}) \ggg s_2.(ID_{uv}, distance_{uv})$.

Hence $s_3.(ID_{uv}, distance_{uv}) \ggg s_3.(ID_u, distance_u)$, which contradicts the earlier assertion. ∎

6.3.2 ID Overrunning Properties

We now show that nodes must "learn" about "high" IDs existing in the network within 2δ time—the smallest ID in the network after time $t + 2\delta$ is at least as large as the highest ID at time t. In this sense, high IDs "overrun" lower IDs.

Lemma 6.14 *Let* $\text{Dist}(u, v) = d$. *For any state* s, *there exists a state* s' *following* s *such that* $s'.now \leq s.now + 2d$ *and* $s'.(ID_v, distance_v) \ggg s.(ID_u, (distance_u + d))$.

Proof. By induction on d.

58

First, let $d = 0$. u is the only node a distance of 0 from itself. Substituting $d = 0$ in the statement, it can be seen to be trivially true ($s' = s$).

Now for the inductive step, for any node v such that $\mathsf{Dist}(u, v) = k$, assume that there exists s' such that $s'.now \le s.now + 2k$ and $s'.(ID_v, distance_v) \gg s.(ID_u, (distance_u + k))$. Consider a node w such that $\mathsf{Dist}(u, w) = k + 1$. We need to show that there exists s'' such that $s''.now \le s.now + 2(k + 1)$ and $s''.(ID_w, distance_w) \gg s.(ID_u, (distance_u + k + 1))$.

Node w must then have a neighbor v such that $\mathsf{Dist}(u, v) = k$. By the inductive hypothesis, there exists a s' such that $s'.now \le s.now + 2k$ and $s'.(ID_v, distance_v) \gg s.(ID_u, (distance_u + k))$.

Now there must exist a step $(s_1, \mathrm{COPY}_{wv}, s_2)$ at some time after $(s.now + 2k)$ and upto $(s.now + 2k + 1)$, since our adversary must allow w to execute every action in every unit of time. Since $s_1.now > s'.now$, by Lemma 6.5, $s_1.(ID_v, distance_v) \gg s'.(ID_v, distance_v)$. Hence $s_1.(ID_v, distance_v) \gg s.(ID_u, distance_u + k)$. Hence $s_2.(ID_{wv}, distance_{wv}) \gg s.(ID_u, (distance_u + k))$.

There must exist another step $(s_3, \mathrm{MAXIMIZE\text{-}PRIORITY}_w, s'')$ at some time after $(s.now + 2k+1)$ and upto $(s.now + 2k + 2)$. By Claim 6.7, $s_3.(ID_{wv}, distance_{wv}) \gg s_2.(ID_{wv}, distance_{wv})$. After statement [E] of $\mathrm{MAXIMIZE\text{-}PRIORITY}_w$, $(ID_{wl}, distance_{wl}) \gg (ID_{wv}, distance_{wv})$. Either statement [G] or [H] must be executed. If [G] is executed, $s''.(ID_w, distance_w)$ $= s_3.(ID_{wl}, (distance_{wl} + 1)) \gg s_3.(ID_{wv}, (distance_{wv} + 1)) \gg s_2.(ID_{wv}, (distance_{wv} + 1)) \gg s.(ID_u, (distance_u + k + 1))$. Hence there exists s'' such that $s''.now \le (s.now + 2k + 2)$ and $s''.(ID_w, distance_w) \gg s.(ID_u, (distance_u + k + 1))$.

If [H] is executed, let the intermediate value of ID_u after executing [F] be I. Then, since [H] is executed, $s''.(ID_w, distance_w) = (I, 0)$

$$\gg (I, s_3.distance_w)$$
$$\gg s_3.(ID_{wl}, distance_{wl})$$
$$\gg s_3.(ID_{wv}, distance_{wv})$$
$$\gg s_2.(ID_{wv}, distance_{wv})$$
$$\gg s.(ID_u, distance_u + k)$$
$$\gg s.(ID_u, distance_u + k + 1).$$

∎

Corollary 6.15 *Let* $\mathsf{Dist}(u,v) = d$. *For any state* s, *for all states* s' *such that* $s'.now \geq (s.now + 2d)$, $s'.(ID_v, distance_v) \gg s.(ID_u, distance_u + d)$.

Proof. Immediate from Lemma 6.5 and Lemma 6.14. ∎

Corollary 6.16 *Let* $\mathsf{Dist}(u,v) = d$. *For any* s, *there exists* s' *following* s *in* α *such that* $s'.now \leq s.now + 2d$ *and* $s'.ID_v \succeq s.ID_u$. ∎

Definition 6.17 (MAXID) *Given a state* $s \in \mathcal{C}^=$, $s.\mathsf{MAXID} \triangleq \max(\xi(s))$.

Corollary 6.18 *For any* s, *there exists* s' *following* s *in* α *such that* $s'.now \leq s.now + 2\delta$ *and* $\forall u \in V$, $s'.ID_u \succeq s.\mathsf{MAXID}$. *For all* s'' *such that* $s''.now > s.now + 2\delta$, $s''.ID_u \succeq s.\mathsf{MAXID}$.
∎

6.3.3 Candidate Root Properties

We first state a very important property of the set $\Phi(s)$. In effect, the ID of each root in $\Phi(s)$ is a prefix of the highest such ID.

Observation 6.19 *For any* s *and any* $u, v \in \Phi(s)$,

1. $\mathrm{IDLENGTH}(s.ID_u) < \mathrm{IDLENGTH}(s.ID_v) \implies s.ID_u \overset{w}{\prec} s.ID_v$.

2. IDLENGTH$(s.ID_u)$ = IDLENGTH$(s.ID_v)$ $\implies$ $s.ID_u = s.ID_v$.

Proof. If $u, v \in \Phi(s)$, by the definition of $\Phi(s)$, it cannot be the case that $ID_u \stackrel{s}{\prec} ID_v$ or $ID_u \stackrel{s}{\succ} ID_v$. Hence $ID_u = ID_v$, or $ID_u \stackrel{w}{\prec} ID_v$, or $ID_u \stackrel{w}{\succ} ID_v$. Hence IDLENGTH$(s.ID_u) <$ IDLENGTH$(s.ID_v)$ must imply $ID_u \stackrel{w}{\prec} ID_v$, and IDLENGTH$(s.ID_u)$ = IDLENGTH$(s.ID_v)$ must imply $ID_u = ID_v$. ∎

Consider any root node r. The following lemma states that as long as r stays a root, its ID can only change by *extension* (only by invoking the call Append-Entry() through actions MAXIMIZE-PRIORITY$_r$ or EXTEND-ID$_r$).

Lemma 6.20 *Let s, s' be any states such that s' follows s in α.*
If $r \in (\rho(s) \cap \rho(s'))$, $s.ID_r \stackrel{w}{\preceq} s'.ID_r$.

Proof. Consider a node $r \in (\rho(s) \cap \rho(s'))$. Let α_1 be the execution fragment $sa_1s_1 \ldots a_is'$. If there exists a state $s_i \in \alpha_1$ such that $r \notin \rho(s_i)$, then $r \notin \rho(s')$ by Lemma 6.13. Hence $r \in \rho(s_i)$ for every state s_i in α_1.

Hence for every step $(s_i, a, (\Omega, \Sigma, P))$ in α_1, for every state s_j in Ω, $s_i.parent_r = s_j.parent_r =$ nil. Thus in action a, statement $[\mathbf{G}]$ of MAXIMIZE-PRIORITY$_r$ could not have been executed. Hence the only way ID_r can change is through the call to Append-Entry(), made by $[\mathbf{F}]$ of MAXIMIZE-PRIORITY$_r$ or by EXTEND-ID$_r$. By Proposition 4.4(1), for every such s_i and s_j, $s_i.ID_r \stackrel{w}{\preceq} s_j.ID_r$. By transitivity of $\stackrel{w}{\preceq}$, it follows that $s.ID_r \stackrel{w}{\preceq} s'.ID_r$. ∎

The following is a crucial property of our algorithm. To ensure fast progress, we want to ensure that if a root r_1 has an ID that is smaller than that of another root r_2, then the relationship will stay that way, even if the two roots never communicate directly. We can ensure this only if r_2's ID is higher *in the strong sense*.

Lemma 6.21 *For all s, s' such that s' follows s in α,*
if $r_1, r_2 \in (\rho(s) \cap \rho(s'))$, $s.(ID_{r1} \stackrel{s}{\prec} ID_{r2}) \implies s'.(ID_{r1} \stackrel{s}{\prec} ID_{r2})$.

61

Proof. Immediate from Lemma 6.20 and Proposition 4.4(5). ∎

We now show that the set Φ is the set of roots that have a chance of "surviving" - a root not in this set cannot be the root of the final spanning tree, and will definitely be overrun by some other tree. We now have a "competition" between roots in the forest. The "winner" of the competition will be the root of the eventual spanning tree. The set Φ is the set of roots still in the fray; all other roots have "lost" and will be overrun. All roots change their IDs only by extension (unless they cease to be a root), and by changing their ID they may lose their membership in Φ.

Lemma 6.22 *For all s, s' such that s' follows s in α, $\Phi(s') \subseteq \Phi(s)$.*

Proof. Suppose not. Then there exists a node r such that $r \in \Phi(s')$ but $r \notin \Phi(s)$. Since $r \in \rho(s')$, by Lemma 6.13 $r \in \rho(s)$. By the definition of $\Phi(s)$, there exists some node $q \in \Phi(s)$ such that $s.ID_r \stackrel{s}{\prec} s.ID_q$. But by Corollary 6.6, $s.ID_q \preceq s'.ID_q$. Hence by Proposition 4.4(3), $s.ID_r \stackrel{s}{\prec} s'.ID_q$. By Lemma 6.20, $s.ID_r \stackrel{w}{\preceq} s'.ID_r$. Applying Proposition 4.4(4), $s'.ID_r \stackrel{s}{\prec} s'.ID_q$. Thus $r \notin \Phi(s')$, contradicting our earlier supposition. ∎

Proposition 6.23 states that if in some state s the set Φ has just one member, a state s' defining a spanning tree is reached within 2δ time.

Proposition 6.23 $\mathcal{G} \stackrel{2\delta}{\longrightarrow} \mathcal{ST}$

Proof. Let s be a state in $\mathcal{G}$. By Corollary 6.18, there exists a state s' following s such that $s'.now \leq s.now + 2\delta$ and for all $u \in V$, $s'.ID_u \succeq s.\mathsf{MAXID}$.

We have $|\Phi(s)| = 1$; therefore, a unique node r has the maximum ID in s. Consider any node $q \neq r$ in $\rho(s)$. By definition of $\Phi(s)$, $s.ID_q \stackrel{s}{\prec} s.ID_r$. Now if $q \in \rho(s')$, by Lemma 6.20, $s.ID_q \stackrel{w}{\preceq} s'.ID_q$, which implies $s'.ID_q \stackrel{s}{\prec} s.ID_r$ by Proposition 4.4(4). But this contradicts our choice of s', since s' was chosen such that $s'.ID_q \succeq s.ID_r$. Thus any node $q \neq r$ in $\rho(s)$ cannot be in $\rho(s')$. Since $\rho(s') \subseteq \rho(s)$ by Lemma 6.13, it follows that $\rho(s') = \{r\}$, and so $s' \in \mathcal{ST}$. ∎

Proposition 6.24 $\mathcal{F} \xrightarrow{2\delta} \mathcal{C}^= \cup \mathcal{C}^1$

Proof. Let $s \in \mathcal{F}$. Consider any execution $\alpha = sa_1 s_1 a_2 s_2 \ldots$; let $\mu = s.\mathsf{MAXID}$. By Corollary 6.18, there exists a state s_k following s in α such that $s_k.now \leq s.now + 2\delta$, and for all $u \in V$, $s_k.ID_u \succeq \mu$. Consider the execution prefix $\alpha_1 = sa_1 s_1 a_2 \ldots s_k$ of α. We show that there must exist some state s' in α_1 such that $s' \in \mathcal{C}^= \cup \mathcal{C}^1$. For all u in $\Phi(s_k)$, $s_k.ID_u \succeq \mu$. Consider the following mutually exhaustive possibilities for $\Phi(s_k)$:

Case 1 For all $u \in \Phi(s_k)$, $s_k.ID_u = \mu$.

Then $s_k \in \mathcal{C}^=$, and we are done.

Case 2 For some $u \in \Phi(s_k)$, $s_k.ID_u \overset{s}{\succ} \mu$.

Since $\Phi(s_k) \subseteq \Phi(s)$, by Lemma 6.22, $s_k.ID_u \overset{s}{\succ} \mu$ for some $u \in \Phi(s)$. Since each step in α changes at most one ID, and since $s.ID_x \preceq \mu$ for all $x \in \Phi(s)$, there must exist some state s' in α such that there is *exactly* one node $v \in \Phi(s)$ for which $s'.ID_v \overset{s}{\succ} \mu$. Since $\Phi(s') \subseteq \Phi(s)$ by Lemma 6.22, v is the only node in $\Phi(s')$ such that $s'.ID_v \overset{s}{\succ} \mu$. Hence $ID_v = \max_{w \in \Phi(s')}(ID_w)$, which implies that $v \in \Phi_{l_{max}}(s')$. There cannot exist another node $w \in \Phi_{l_{max}}(s')$, since that would imply that $s'.ID_w = s'.ID_v$, which would violate our assumption that v is the only node in $\Phi(s')$ such that $s'.ID_v \overset{s}{\succ} \mu$. Hence $|\Phi_{l_{max}}(s')| = 1$, and so $s' \in \mathcal{C}^1$.

Case 3 $ID_u \overset{w}{\succeq} \mu$ for all $u \in \Phi(s_k)$, and there is at least one node $u \in \Phi(s_k)$ such that $s_k.ID_u \overset{w}{\succ} \mu$.

Since $\Phi(s_k) \subseteq \Phi(s)$, by Lemma 6.22, there is at least one node $u \in \Phi(s)$ such that $s_k.ID_u \overset{w}{\succ} \mu$. Since each step in α changes at most one ID, and since $s.ID_x \preceq \mu$ for all $x \in \Phi(s)$, there must exist some state s' in α such that there is *exactly* one node $v \in \Phi(s)$ for which $s'.ID_v \overset{w}{\succ} \mu$. There cannot exist a node $w \in \Phi(s)$ such that $s'.ID_w \overset{s}{\succ} \mu$, since that would imply by Corollary 6.6 that $s_k.ID_w \succeq s'.ID_w$ and hence by Proposition 4.4(3) that $s_k.ID_w \overset{s}{\succ} \mu$, which contradicts our assumption that $s_k.ID_u \overset{w}{\succeq} \mu$ for all $u \in \Phi(s_k)$. Thus

for all u other than v in $\Phi(s)$, $s'.ID_u \stackrel{w}{\preceq} \mu$. Thus $\Phi_{l_{max}}(s') = \{u\}$; hence $|\Phi_{l_{max}}(s')| = 1$ and $s' \in \mathcal{C}^1$. ∎

6.4 The ID-forcing Proposition

In this section we prove the statement $\mathcal{C}^1 \xrightarrow[0.11]{2\delta} \mathcal{G}$, i.e., starting from a state in which there is only one candidate of maximal ID length, within 2δ time, with probability at least 0.11, we reach a "good" state—a state in which there is just one candidate. This is a substantial progress property, since if a state is "good" then within 2δ additional time we reach a state defining a spanning tree.

Let s be a state in $\mathcal{C}^1$, and let $\overline{H}$ be the execution automaton $H(\mathsf{RSST}, \mathcal{A}, s)$. Let β' be a maximal execution of $\overline{H}$, and let $\beta = \beta'{\downarrow} = sa_1s_1a_2s_2 \ldots$ be the corresponding execution of RSST. Let l_{min} denote $\min_{u \in \Phi(s)}(\mathrm{IDLENGTH}(s.ID_u))$, and let l_{max} be defined analogously. Thus all nodes in $\Phi(s)$ have ID lengths between l_{min} and l_{max}. Let $\mu = s.\mathsf{MAXID}$, and let r be the unique element of $\Phi(s)$ such that $s.ID_r = \mu$. (Since $s \in \mathcal{C}^1$, r is unique.) Thus r is the unique candidate root in s having the maximum ID length l_{max}.

By the ID overrunning property, Corollary 6.18, there exists a state s_k following s in β such that $s_k.now \leq s.now + 2\delta$ and for all $u \in V$, $s_k.ID_u \succeq \mu$. Let s_k be the *first* such state in β. Let β_1 be the execution prefix $sa_1s_1a_2s_2 \ldots s_k$.

We will use these definitions of s, s_k, β, β_1, l_{min}, l_{max}, μ, and r throughout the rest of this section.

We first give some basic definitions and observations related to these definitions.

Definition 6.25 (Competitive and dominant nodes) Let $\overline{H}$, s, s_k, μ, β and β_1 be as defined above, and let $i \leq l_{max}$. Then,

- Node u is *competitive at the ith position in β*, if there exists $s' \in \beta_1$ such that $u \in \rho(s')$ and $s'.ID_u[1..i] = \mu[1..i]$.

64

- Node u is *dominant at the ith position in β*, if there exists $s' \in \beta_1$ such that $u \in \rho(s')$, $s'.ID_u[1..(i-1)] = \mu[1..(i-1)]$, and $s'.ID_u[i] > \mu[i]$. Node u is *dominant before the ith position in β*, if there exists a $j < i$ such that u is dominant at the jth position.

We now state some observations arising from the above definitions. The first property states that competitiveness and dominance of a node at a particular position are mutually exclusive:

Claim 6.26 *A node u cannot be both competitive and dominant at the ith position, for any i.*

Proof. Suppose u is competitive and dominant at the ith position in β. Since it is competitive, there exists $s' \in \beta$ such that $u \in \rho(s')$ and $s'.ID_u[i] = \mu[i]$. Since it is dominant, there exists $s'' \in \beta$ such that $u \in \rho(s'')$ and $s''.ID_u[i] > \mu[i]$. Clearly, $s'.ID_u[i] \neq s''.ID_u[i]$.

Now s' must either precede or follow s'' in β. If s' precedes s'', Lemma 6.20 implies that $s'.ID_u \overset{w}{\preceq} s''.ID_u$, which implies $s'.ID_u[i] = s''.ID_u[i]$, which is a contradiction. Similarly, the other case, s' follows s'', leads to the same contradiction. ∎

Claim 6.27 *If a node u is either competitive or dominant at the ith position in β, it is competitive at the jth position for all $j < i$.*

Proof. Straightforward from Definition 6.25. ∎

Claim 6.28 *If a node u dominant at the ith position in β, it cannot be competitive at the jth position for any $j \geq i$.*

Proof. Follows directly from Claims 6.26 and 6.27. ∎

Claim 6.29 *Any $u \in \Phi(s)$ is competitive at the l_{min}^{th} position in β.*

Proof. By the definition of μ, and by Observation 6.19, $s.ID_u \overset{w}{\preceq} \mu$, and further, IDLENGTH$(s.ID_u) > l_{min}$. Hence $s.ID_u[1..l_{min}] = \mu[1..l_{min}]$. ∎

65

Corollary 6.30 *No node $u \in \Phi(s)$ is dominant before the $(l_{min} + 1)^{th}$ position in β.*

Proof. Follows directly from Claims 6.28 and 6.29. ∎

Definition 6.31 (Competitive and dominant executions) Let $\overline{H}$, s, s_k, μ, β and β_1 be as defined above. Then,

- Execution β is *competitive at the ith position* if no node is dominant before the $(i + 1)^{th}$ position.

- Execution β is *dominant at the ith position* if no node is dominant before the ith position and there exists $u \in \Phi(s)$ such that u is dominant at the ith position. ∎

Claim 6.32 *An execution β cannot be both competitive and dominant at the ith position, for any i.*

Proof. Follows directly from Definition 6.31. ∎

Claim 6.33 *Let $i < l_{max}$. If β is competitive at the ith position, it is either competitive or dominant at the $(i + 1)^{th}$ position.*

Proof. Since β is competitive at the ith position, no node is dominant before the $(i + 1)^{th}$ position. If some node is dominant at the $(i+1)^{th}$ position, β is dominant at the $(i+1)^{th}$ position. Otherwise, no node is dominant before the $(i + 2)^{th}$ position, and hence β is competitive at the $(i + 1)^{th}$ position. ∎

Having described competitive and dominant executions, we now define the corresponding *events* of $\overline{H}$.

Definition 6.34 (Competitive and dominant events) Let $\overline{H}$, s, and s_k, be as defined above. Then,

- The event $e_C^{[i]}$, "competitiveness at position i," is defined as

$$e_C^{[i]} \triangleq \{\beta{\uparrow} \in \Omega_{\overline{H}} \mid \beta \text{ is competitive at the } i^{th} \text{ position}\}$$

- The event $e_C^{[i,j]}$ consists of those executions in $e_C^{[i]}$ in which exactly j nodes in $\Phi(s)$ are competitive at the ith position.

- The event $e_D^{[i]}$, "dominance at position i," is defined as

$$e_D^{[i]} \triangleq \{\beta\uparrow \in \Omega_{\overline{H}} \mid \beta \text{ is dominant at the } i^{th} \text{ position}\}$$

- The event $e_{\mathcal{G}}$ is defined as a subset of the set of executions in which a state in $\mathcal{G}$ is reached within time 2δ; in particular,

$$e_{\mathcal{G}} \triangleq \{\beta\uparrow \in \Omega_{\overline{H}} \mid s_k(\beta) \in \mathcal{G}\}$$

We now state some important properties of events.

Claim 6.35 $e_C^{[i]} = \bigcup_{j=1}^{n} e_C^{[i,j]}$.

Proof. From the definitions (recall that n is the size of the network). ∎

Claim 6.36 *For any* $i \leq l_{max}$, $e_C^{[i]} \cap e_D^{[i]} = \phi$.

Proof. Follows from Claim 6.32. ∎

Claim 6.37 *For any* $i \leq (l_{max} - 1)$, $e_C^{[i+1]}, e_D^{[i+1]} \subseteq e_C^{[i]}$.

Proof. Follows from Definitions 6.34 and 6.31. ∎

Claim 6.38 *For any* $i < (l_{max} - 1)$, $e_C^{[i]} = e_D^{[i+1]} \cup e_C^{[i+1]}$.

Proof. By Claim 6.37, $e_D^{[i+1]} \cup e_C^{[i+1]} \subseteq e_C^{[i]}$. By Claim 6.33, $e_C^{[i]} \subseteq e_D^{[i+1]} \cup e_C^{[i+1]}$. Hence follows. ∎

Claim 6.39 $\Omega_{\overline{H}} = e_C^{[l_{min}]}$.

Proof. Consider any execution $\beta \upharpoonright \in \Omega_{\overline{H}}$. From Corollary 6.30, it follows that β is competitive at the l_{min}^{th} position. ∎

Claim 6.40 $\Omega_{\overline{H}} = e_D^{[l_{min}+1]} \cup e_D^{[l_{min}+2]} \cup e_D^{[l_{min}+3]} \cup \ldots \cup e_D^{[l_{max}-1]} \cup e_C^{[l_{max}-1]}$.

Proof. We have,

$$
\begin{aligned}
\Omega_{\overline{H}} &= e_C^{[l_{min}]} & \text{(Claim 6.39)} \\
&= e_D^{[l_{min}+1]} \cup e_C^{[l_{min}+1]} & \text{(Claim 6.38)} \\
&= e_D^{[l_{min}+1]} \cup e_D^{[l_{min}+2]} \cup e_C^{[l_{min}+2]} & \text{(Applying Claim 6.38 again)} \\
&= e_D^{[l_{min}+1]} \cup e_D^{[l_{min}+2]} \cup \ldots \cup e_D^{[l_{max}-1]} \cup e_C^{[l_{max}-1]} & \text{(Inductively applying Claim 6.38)}
\end{aligned}
$$
∎

Note that Claim 6.40 defines a *partition* of $\Omega_{\overline{H}}$.

Definition 6.41 Node u *flips at the* i^{th} *position in* β, if in β_1 there exists a step (s', a, s'') such that in a, u makes a call to Append-Entry which appends an entry to ID_u at the ith position.

Lemma 6.42 *Let* $\beta \in e_C^{[i]}$. *For any* $u \in \Phi(s)$, *if* IDLENGTH$(s.ID_u) \leq i$, *and if* u *is competitive at the* i^{th} *position in* β, *then* u *flips at the* $(i+1)^{th}$ *position in* β.

Proof. Consider $\beta_1 = s a_1 s_1 a_2 \ldots s_k$. Since u is competitive at the ith position in β, there exists a $s_i \in \beta$ such that $u \in \rho(s_i)$ and $s_i.ID_u[1..i] = \mu[1..i]$. Also, by the definition of s_k, $s_k.ID_u \succeq \mu$. Now, by Lemma 6.20, ID_u can only change by extension in $s a_1 s_1 a_2 \ldots s_k$, so we can choose s_i such that $s_i.ID_u = \mu[1..i]$.

Consider the suffix of the execution that starts with s_i. The only way ID_u can change between s_i and s_k is by executing calls to Append-Entry or by executing statement **[G]**. If Append-Entry is performed first, an entry is appended at the $(i+1)^{th}$ position, so we are done. If **[G]** is executed, there exists a node l such that $ID_{ul} \succ ID_u$. By Claim 6.3, $ID_{ul} = ID_l$ for some preceding state. Since $\beta \in e_C^{[i]}$, l is not dominant before the $(i+1)^{th}$ position, and hence $ID_l[1..i] = \mu[1..i]$. Hence $ID_u \stackrel{w}{\prec} ID_u l$, and so the call to Append-Entry in **[F]** must have been executed first, in which case u would have flipped at the $(i+1)^{th}$ position. ∎

Lemma 6.43 *For any* $i \leq l_{max}$, $e_C^{[i,1]} \subseteq e_{\mathcal{G}}$.

Proof. If $\beta \in e_C^{[i,1]}$, no node is dominant before the $(i+1)^{th}$ position, so for any $u \in \Phi(s_k)$, $s_k.ID_u[1..i] = \mu[1..i]$. But then any such node is competitive at the ith position, and there is only one such node, since $\beta \in e_C^{[i,1]}$. Hence $|\Phi(s_k)| = 1$, and so $s_k \in \mathcal{G}$. ∎

We now list, without proof, some basic results of conditional probability:

Proposition 6.44 *Let* A, A_i, B, B_i, *and* X *be events on a sample space. Then,*

1. *If* $A = \bigcup_{i=1}^{k} A_i$, *then* $P(X \mid A) \geq \min_i P(X \mid A_i)$.

2. *If* $A \subseteq \bigcup_{i=1}^{k} A_i$, *then* $P(X \mid A) \geq \min_i P(X \mid A \cap A_i)$.

3. *Let* $\bigcup_{i=1}^{k} A_i \subseteq A$. *If* $P((\bigcup_{i=1}^{k} A_i) \mid A) = p$, *and if* $P(X \mid A_i) = p_i$, *then* $P(X \mid A) \geq p \times \min_i\{p_i\}$. ∎

Lemma 6.45 *For any* i *such that* $(l_{min} + 1) \leq i \leq l_{max}$ *and any* $j \geq 1$, $P(e_{\mathcal{G}} \mid e_D^{[i]} \cap e_C^{[i-1,j]}) \geq 1/2$.

Proof. Consider any execution $\beta \in e_D^{[i]} \cap e_C^{[i-1,j]}$. Since $\beta \in e_C^{[i-1,j]}$, there are j nodes competitive at the $(i-1)^{th}$ position. Of these j nodes, there are exactly $|\Phi_{>(i-1)}(s)|$ nodes u such that IDLENGTH$(s.ID_u) > (i-1)$, and consequently $k = j - |\Phi_{>(i-1)}(s)|$ nodes such that IDLENGTH$(s.ID_u) \leq (i-1)$. Thus by Lemma 6.42, each of these k nodes must flip at the ith position. Hence , $_{AM}^k$ describes the sample space corresponding to these k flips. If $\beta \in e_D^{[i]}$, there exists a flip higher than $\mu[i]$. If exactly one of these flips is the highest, then $\beta \in e_{\mathcal{G}}$. Thus,

$$P(e_{\mathcal{G}} \mid e_D^{[i]} \cap e_C^{[i-1,j]}) \geq P_{\Gamma^k}(\text{UNIQH} \mid (\textit{Highest} > \mu[i])) \geq 1/2,$$

by Theorem 4.2. ∎

Theorem 6.46 *For any* i *such that* $(l_{min} + 1) \leq i \leq l_{max}$, $P(e_{\mathcal{G}} \mid e_D^{[i]}) \geq 1/2$.

Proof. We have $e_D^{[i]} \subseteq e_C^{[i-1]}$ by Claim 6.38. Thus by Claim 6.35,

$$e_D^{[i]} \subseteq \bigcup_{j=1}^{n} e_C^{[i-1,j]}$$

which implies, by Proposition 6.44(2), that

$$P(e_{\mathcal{G}} \mid e_D^{[i]}) \;\geq\; \min_j P(e_{\mathcal{G}} \mid e_D^{[i]} \cap e_C^{[i-1,j]}).$$

Since by Lemma 6.45 $P(e_{\mathcal{G}} \mid e_D^{[i]} \cap e_C^{[i-1,j]}) \geq 1/2$ for all j, it follows that $P(e_{\mathcal{G}} \mid e_D^{[i]}) \geq 1/2$. ∎

Lemma 6.47 *For any $j \geq 1$, $P((e_D^{[l_{max}]} \cup e_C^{[l_{max},1]}) \mid e_C^{[l_{max}-1,j]}) \geq 0.22$.*

Proof. Consider any $\beta \in e_C^{[l_{max}-1,j]}$. There are j nodes competitive at the $(l_{max}-1)^{th}$ position; of these, IDLENGTH$(s.ID_r) > l_{max} - 1$ for exactly one node r, and thus IDLENGTH$(s.ID_u) \leq l_{max}-1$ for exactly $(j-1)$ nodes u. Thus by Lemma 6.42 these $(j-1)$ nodes must flip at the l_{max}^{th} position in β, and the sample space $,_{AM}^{j-1}$ describes these flips. The event $e_D^{[l_{max}]}$ is equivalent to the event $(Highest > \mu[l_{max}])$. The event $e_C^{[l_{max},1]}$ is equivalent to the event $(Highest < \mu[l_{max}])$, since one node r is already known to be competitive at the l_{max}^{th} position. Thus,

$$P((e_D^{[l_{max}]} \cup e_C^{[l_{max},1]}) \mid e_C^{[l_{max}-1,j]}) = \; P_{\Gamma^{j-1}}(Highest \neq \mu[l_{max}]) \;\geq 0.22$$

by Theorem 4.3. ∎

Theorem 6.48 $P(e_{\mathcal{G}} \mid e_C^{[l_{max}-1]}) \geq 0.11$

Proof. Consider the event $e_C^{[l_{max}-1,j]}$.

By Lemma 6.47, $P((e_D^{[l_{max}]} \cup e_C^{[l_{max},1]}) \mid e_C^{[l_{max}-1,j]}) \;\geq\; 0.22$. Thus, we have

$$P((e_D^{[l_{max}]} \cap e_C^{[l_{max}-1,j]}) \cup (e_C^{[l_{max},1]} \cap e_C^{[l_{max}-1,j]}) \mid e_C^{[l_{max}-1,j]}) \;\geq\; 0.22$$

Also, by Lemma 6.45,

$$P(e_{\mathcal{G}} \mid e_D^{[l_{max}]} \cap e_C^{[l_{max}-1,j]}) \geq 1/2$$

and by Lemma 6.43,

$$P(e_{\mathcal{G}} \mid (e_C^{[l_{max},1]} \cap e_C^{[l_{max}-1,j]})) = 1.$$

Hence applying Proposition 6.44(3), we have

$$P(e_{\mathcal{G}} \mid e_C^{[l_{max}-1,j]}) \geq (0.22)(1/2) = 0.11.$$

Now by Claim 6.35, $e_C^{[l_{max}-1]} = \bigcup_{j=1}^{n} e_C^{[l_{max}-1,j]}$. Thus by Proposition 6.44(1),

$$P(e_{\mathcal{G}} \mid e_C^{[l_{max}-1]}) \geq \min_{j} P(e_{\mathcal{G}} \mid e_C^{[l_{max}-1,j]}) \geq 0.11.$$

■

Proposition 6.49 $P(e_{\mathcal{G}} \mid \Omega_{\overline{H}}) \geq 0.11$, or equivalently, $\mathcal{C}^1 \xrightarrow[0.11]{2\delta} \mathcal{G}$.

Proof. By Claim 6.40, $\Omega_{\overline{H}} = e_D^{[l_{min}+1]} \cup e_D^{[l_{min}+2]} \cup e_D^{[l_{min}+3]} \cup \ldots \cup e_D^{[l_{max}-1]} \cup e_C^{[l_{max}-1]}$. By Theorem 6.46, $P(e_{\mathcal{G}} \mid e_D^{[i]}) \geq 1/2$, and by Theorem 6.48, $P(e_{\mathcal{G}} \mid e_C^{[l_{max}-1]}) \geq 0.11$. Hence applying Proposition 6.44(1),

$$\begin{aligned}
P(e_{\mathcal{G}} \mid \Omega_{\overline{H}}) &= P(e_{\mathcal{G}} \mid e_D^{[l_{min}+1]} \cup e_D^{[l_{min}+2]} \cup \ldots \cup e_D^{[l_{max}-1]} \cup e_C^{[l_{max}-1]}) \\
&\geq \min(1/2, 1/2, \ldots, 1/2, 0.11) \\
&= 0.11
\end{aligned}$$

■

Chapter 7

Correctness and Complexity Proof: Part 2 – The Coloring Algorithm

In this section we prove the Tree Detection Proposition, $\mathcal{C}^= \xrightarrow[2/9]{77\delta+36} \mathcal{C}^1$ (Proposition 7.80). Thus, starting from a state in $\mathcal{C}^=$, within time $77\delta + 36$, with probability at least $2/9$, we reach a state in which only one candidate has the maximal ID length. This is the "tree detection" property—if, in some state, all root nodes in the network have equal IDs, then, because of the coloring, the competition makes "progress" within expected $O(\delta)$ time.

The overall strategy of the proof is as follows: We first show, in Lemma 7.1, that starting from a state $s \in \mathcal{C}^=$, any execution fragment α must remain in $\mathcal{C}^=$ until a state in $\mathcal{C}^1$ is reached. Thus, to show the partial progress properties of the coloring algorithm, we consider an execution fragment α_1 in $\mathcal{C}^=$. Next, in Section 7.1, we show that within time $2\delta + 1$ in α_1, a state defining a "stable forest" is reached. (We denote the set of states defining a stable forest by $\mathcal{C}^=_{\overline{SF}}$.) Let α_2 be any execution fragment in $\mathcal{C}^=_{\overline{SF}}$. The graph of *parent* pointers remains fixed in $\mathcal{C}^=_{\overline{SF}}$; the network can thus be visualized as a collection of "fixed" trees over which the coloring algorithm runs.

When a state in $\mathcal{C}^=_{\overline{SF}}$ is reached, the coloring variables (i.e., *color*, *mode*) may be in an inconsistent state—normal "broadcast" and "echo" waves (cf. Section 5.1) may not be able to

commence immediately. Section 7.2 shows that within time $17\delta + 7$ in α_2, a state is reached in which the coloring variables become consistent. ($\mathcal{C}_{\overline{W}C}^{\overline{=}}$ is the state set consisting of such states.) The coloring algorithm can proceed normally in any execution fragment α_3 in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$.

Let α_3 be a fragment in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$. For any tree T_r, α_3 can be partitioned into *coloring epochs* for T_r. In each coloring epoch γ in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$, the root color is propagated to all nodes in T_r (through a "broadcast wave"), and the root waits for all nodes in its tree to echo before choosing a new color and initiating the next coloring epoch. If a node with a non-zero color in some tree T notices that a neighbor has a non-zero color different from its own color, it sets *other-trees* to true, and this information is propagated to its root. (It is "piggy-backed" on the "echo wave"; its ancestors successively set their *other-trees* to true while echoing.) After a root sets *other-trees* to true, it extends its ID, thus reaching a state in $\mathcal{C}^1$.

As discussed in Section 5.1, when a node receives a new non-zero color it *waits* until 1) it has observed a non-zero color for each of its neighbors, and 2) each neighbor has observed its own color. Section 7.2.2 and Lemma 7.61 show that a node cannot be "blocked" by its neighbors in this fashion for more than $10\delta + 5$ time. Based on this result, a coloring epoch cannot last more than $13\delta + 6$ time. (Note that the individual node "waits" are not dependent on each other; they can overlap.)

Each coloring epoch in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$ gives a tree at least one "opportunity" to detect neighboring trees, and each epoch lasts at most $13\delta + 6$ time. Section 7.3 formalizes this notion. If T and $\overline{T}$ are neighboring trees, we show that starting from a state in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$, at least one of the two trees must detect the existence of the other within time $58\delta + 28$, with probability at least $2/9$. When this information is conveyed to the root of the "noticing" tree shortly thereafter, that root extends its ID, and a state in $\mathcal{C}^1$ is reached. Since the total time elapsed starting from a state in $\mathcal{C}^=$ would then be $77\delta + 36$, the Tree Detection Proposition (Proposition 7.80) follows.

We now proceed with the details of the proof.

Lemma 7.1 *Let $\alpha = s_0 a_1 s_1 \ldots s_k$ be an execution fragment of RSST, and let $s_0 \in \mathcal{C}^=$. Then, unless a state in $\mathcal{C}^1$ is reached in α, the following conditions hold for all states s in α:*

1. $s \in \mathcal{C}^=$

2. $\Phi(s) = \Phi(s_0)$

3. $s.\mathsf{MAXID} = s_0.\mathsf{MAXID}$.

Proof. Consider any step (s, a, s') such that $s \in \mathcal{C}^=$. Since $\mathcal{C}^= \subseteq \mathcal{F}$, by the definition of $\mathcal{F}$, $s.ID_{uv} \preceq s.\mathsf{MAXID}$ for every u,v. Let u be the node executing action a. Then there exist two possibilities:

Case 1 $u \notin \Phi(s)$.

Then $u \notin \Phi(s')$ by Lemma 6.22, and since all other IDs are unchanged, (1) $s' \in \mathcal{C}^=$, (2) $\Phi(s') = \Phi(s)$, and (3) $s'.\mathsf{MAXID} = s.\mathsf{MAXID}$.

Case 2 $u \in \Phi(s)$.

Then u must be in $\rho(s')$, since otherwise u must have executed statement $[\mathbf{G}]$ in MAXIMIZE-PRIORITY$_u$, which would imply that there exists a node l such that $s.ID_{ul} \succ s.ID_u$, which is impossible since $s \in \mathcal{F}$ and $s.ID_u = s.\mathsf{MAXID}$. Thus by Lemma 6.20, $s.ID_u \overset{w}{\preceq} s'.ID_u$. If $s.ID_u = s'.ID_u$, (1) $s' \in \mathcal{C}^=$, (2) $\Phi(s') = \Phi(s)$, and (3) $s'.\mathsf{MAXID} = s.\mathsf{MAXID}$. If $s.ID_u \overset{w}{\prec} s'.ID_u$, then since the IDs of all other roots in Φ are unchanged, $s' \in \mathcal{C}^1$.

By induction on the steps in α, the Lemma follows. ∎

The following definition makes it convenient to describe progress properties of executions starting from a state in $\mathcal{C}^=$. By Lemma 7.1, such an execution must either reach a state in $\mathcal{C}^1$ or remain in $\mathcal{C}^=$. Thus, progress towards a subset U' of $\mathcal{C}^=$ can be described in terms of the following notation:

Definition 7.2 *If U and U' are state sets, then*

$$U \overset{t}{\Longrightarrow} U' \qquad \triangleq \qquad U \overset{t}{\longrightarrow} U' \cup \mathcal{C}^1$$

75

Recall from Section 2 that set U is *closed*, written $U \longrightarrow U \boxdot$, if for any $s \in U$ and any step $(s, a, (\Omega, \Sigma, P))$, $\Omega \subseteq U$. We now give an analogous definition for analyzing the coloring algorithm:

Definition 7.3 $U \Longrightarrow U \boxdot$, *if for any* $s \in U$ *and any step* $(s, a, (\Omega, \Sigma, P))$, $\Omega \subseteq U \cup \mathcal{C}^1$.

Thus if $U \Longrightarrow U \boxdot$, any execution fragment beginning with a state in U remains in U until a state in $\mathcal{C}^1$ is reached.

7.1 Forest Stability

We now define a very important notion, that of a "stable forest." In order for the recoloring algorithm (used to detect other trees) to succeed in $O(\delta)$ expected time, the forest structure must be "stable" while the algorithm is operating, i.e., the *parent* pointers remain fixed. We now precisely define the set $\mathcal{C}^{=}_{SF}$ of states defining a stable forest. We then show that starting from a state in $\mathcal{C}^{=}$, within time $2\delta+1$, unless a state of $\mathcal{C}^1$ is reached, a state defining a stable forest is reached.

Definition 7.4 ($\mathcal{C}^{=}_{SF}$) *The set* $\mathcal{C}^{=}_{SF}$ *("SF" for "Stable Forest") is the set of all states* $s \in \mathcal{C}^{=}$ *for which the following conditions hold for all nodes* u:

1. $s.ID_u = s.\mathsf{MAXID}$,

2. $u \in \rho(s) \Longrightarrow distance_u = 0$, *and*

3. $(parent_u = v) \Longrightarrow$
   - $distance_{uv} = distance_v$,
   - $distance_u = distance_v + 1$, *and*
   - $v = \max_{x \in Nbrs(u)} \{x \mid (ID_x, distance_x) = \max_{w \in Nbrs(u)} (ID_w, distance_w)\}$ ■

Lemma 7.5 *For any step* $(s, a, (\Omega, \Sigma, P))$ *such that* $s \in \mathcal{C}^{=}_{SF}$ *and for any* $s' \in \Omega$, *for all* u *and* v, $s.(parent_u = v) \Longrightarrow s'.(parent_u = v)$.

76

Proof. Let $s' \in \Omega$. The only action a that could change $parent_u$ is MAXIMIZE-PRIORITY$_u$. Since $s \in \mathcal{C}_{\overline{SF}}^{=}$, for any $w \in Nbrs(u)$ such that $w \neq s.parent_u$, $s.(ID_{uw}, distance_{uw}) \ll s.(ID_w, distance_w) = (s.\mathsf{MAXID}, s.distance_w)$. Thus if $v = s.parent_u$, $(ID_{uv}, distance_{uv}) = \max_{x \in Nbrs(u)} (ID_{ux}, distance_{ux})$. Hence l is set to v in [**E**], [**F**] does not change ID_u, and [**G**] ensures that $s'.parent_u = v$. ∎

Lemma 7.6 $\mathcal{C}_{\overline{SF}}^{=} \Longrightarrow \mathcal{C}_{\overline{SF}\boxdot}^{=}$.

Proof. Let $(s, a, (\Omega, \Sigma, P))$ be a step such that $s \in \mathcal{C}_{\overline{SF}}^{=}$. Let $s' \in \Omega$, and let a be performed by u. Since membership in $\mathcal{C}_{\overline{SF}}^{=}$ is determined by the variables ID, $distance$ and $parent$, the actions DETECT-TREES and NEXT-COLOR cannot change membership in $\mathcal{C}_{\overline{SF}}^{=}$. Consider the following remaining possibilities for a:

Case 1 $a = $COPY$_{uv}$.

The only statement of interest is [**A**]; by clause (3) of the definition of $\mathcal{C}_{\overline{SF}}^{=}$, $distance_{u \cdot parent_u}$ must remain unchanged, and $s' \in \mathcal{C}_{\overline{SF}}^{=}$.

Case 2 $a = $MAXIMIZE-PRIORITY$_u$.

If $u \in \rho(s)$, [**H**] is executed, and $s' \in \mathcal{C}_{\overline{SF}}^{=}$. If $u \notin \rho(s)$, let $v = s.parent_u$. Then [**E**] sets l to v, [**F**] has no effect, and [**G**] preserves the values of ID_u, $distance_u$ and $parent_u$. Thus $s' \in \mathcal{C}_{\overline{SF}}^{=}$.

Case 3 $a = $EXTEND-ID$_u$.

If ID_u is extended, then $s' \in \mathcal{C}^1$. ∎

Lemma 7.7 $\mathcal{C}^{=} \overset{2\delta+1}{\Longrightarrow} \mathcal{C}_{\overline{SF}}^{=}$.

Proof. Let $s_0 \in \mathcal{C}^{=}$, and consider any execution fragment $\alpha = s_0 a_1 s_1 a_2 \ldots a_k s_k$ in $\mathcal{C}^{=}$ of duration $\geq 2\delta + 1$. Let the *minimal distance* of node u be defined as

$$\mathcal{D}(u) \triangleq \min_{v \in \Phi(s_0)} \mathsf{Dist}(u, v)$$

We show by induction that for any $i \leq \delta$, there must exist a state s' in α such that $s'.now \leq s_0.now + 2i + 1$, and for each node u such that $\mathcal{D}(u) \leq i$, u satisfies the conditions (1), (2) and (3) in Definition 7.4, for membership in $\mathcal{C}_{\overline{S}F}^{=}$. (Let such a node be called *locally stable*.) Since $\mathcal{D}(u) < \delta$ for all nodes, there must then exist a state s in α such that $s.now < s_0.now + 2\delta + 1$ and all nodes are locally stable, which implies that $s \in \mathcal{C}_{\overline{S}F}^{=}$.

First, let $i = 0$. The only nodes u for which $\mathcal{D}(u) = 0$ are those in the set $\Phi(s_0)$; within 1 time unit, each such node will have executed statement [**H**] of MAXIMIZE-PRIORITY and will have set its *distance* to 0 and will have thus become locally stable.

For the inductive step, let there exist a state s' in α such that $s'.now < s_0.now + 2i + 1$, and each node u for which $\mathcal{D}(u) \leq i$ is locally stable. We show that there exists a state s'' following s' such that $s'' < s_0 + 2(i + 1) + 1$ and each node u such that $\mathcal{D}(u) \leq i + 1$ is locally stable.

The conditions for local stability imply that in state s', $ID_u = s_0.\mathsf{MAXID}$ and $distance_u = \mathcal{D}(u)$ for each node u such that $\mathcal{D}(u) \leq i$. Consider any node u for which $\mathcal{D}(u) = i+1$. Let $\mathcal{N}(u)$ be the set of neighbors w of u for which $\mathcal{D}(w) = i$; $s'.ID_w = s.\mathsf{MAXID}$ and $s'.distance_w = i$ for all such w. There must then exist a state s''' following s' in α such that $s'''.now < s'.now + 1$ and $(ID_{uw}, distance_{uw}) = (s_0.\mathsf{MAXID}, i)$ for all $w \in \mathcal{N}(u)$. Since there must exist a MAXIMIZE-PRIORITY$_u$ step within time 1 after s''', there exists a s'' following s''' in α such that $s''.now \leq s'''.now + 1$ and u is locally stable. Hence the inductive step follows. ∎

7.2 Self-Stabilization of the Coloring Algorithm

As was stated in the previous section, the forest structure must be stable, i.e. the state must be in $\mathcal{C}_{\overline{S}F}^{=}$, while the algorithm is operating. Lemma 7.7 guarantees that starting from any state in $\mathcal{C}^{=}$, a state in $\mathcal{C}_{\overline{S}F}^{=}$ is reached within $2\delta + 1$ time. However, when a state in $\mathcal{C}_{\overline{S}F}^{=}$ is reached, the coloring variables may not be in a consistent state—they may be arbitrarily set, so the broadcast-echo mechanism may not commence immediately. In this section we show

that within time $17\delta + 7$, these variables become consistent, and the coloring algorithm can proceed correctly.

In Definition 7.9, we define a "coloring predicate" $\mathbf{L}(u)$ on individual nodes; if all nodes in a tree T_r satisfy $\mathbf{L}(u)$ and if T_r satisfies another predicate $\mathbf{L}'$, the coloring variables in that tree are consistent. T_r is then said to be "well-colored," and the state set $\mathcal{GT}_r$ ("$\mathcal{GT}$" for "**G**ood **T**ree") is defined as the set of states in $\mathcal{C}_{\overline{SF}}^{=}$ in which T_r is well-colored. $\mathcal{C}_{\overline{WC}}^{=}$ is defined as the set of states in $\mathcal{C}_{\overline{SF}}^{=}$ in which *all* trees are well-colored.

We show that starting from a state s in $\mathcal{C}_{\overline{SF}}^{=}$, unless a state in $\mathcal{C}^1$ is reached, for any tree T_r a state in $\mathcal{GT}_r$ is reached within time $17\delta + 7$ (Lemma 7.65). We do so using the intermediate state set $\mathcal{MT}_r$—the set of states in which T_r is *monocolored*, i.e. all nodes in T_r possess the same color. Section 7.2.1 shows that any tree must get monocolored within time $4\delta + 1$. Section 7.2.2 shows that once a tree is monocolored, it must get well-colored within $13\delta + 6$ additional time.

We first define what it means for coloring variables to be "consistent."

Definition 7.8 (T_r, TREE(v), leaf, root interval, branch, height, BRANCHES(T_r))

- Let $r \in \rho(s)$. A *tree rooted at node r* is the set

$$T_r \triangleq \{u \mid r \text{ is an ancestor of } u.\}$$

- TREE(v), the *tree containing node v*, is defined as the unique tree containing v.

- A *leaf* is a node that is not an ancestor of any other node.

- A sequence of nodes $R = u_1 u_2 \ldots u_k$ is a *root interval* of T_r if $u_1 = r$ and $parent_{u_i} = u_{i-1}$ for every $i > 1$.

- A root interval $B = u_1 u_2 \ldots u_k$ is a *branch* if it terminates in a leaf (i.e., u_k is a leaf).

- The *height* of a tree, written HEIGHT(T_r), is the maximal length of a branch in T_r.

- BRANCHES(T_r) denotes the set of all branches in T_r. ∎

Definition 7.9 (Coloring predicates) Let $v = parent_u$. Then the following *coloring predi-cates* are defined for node u:

- **L1**(u): $(color_u \neq color_v) \Longrightarrow (mode_u = \text{echo})$ **and** $(mode_v = \text{broadcast})$.

- **L2**(u): $mode_u = \text{broadcast} \Longrightarrow$

  - $mode_v = \text{broadcast}$

  - $color_u = color_v$

  - If $w \in \mathsf{Children}_u$,

    $(mode_{uw} = \text{echo}$ **and** $color_{uw} = color_u) \Longrightarrow mode_w = \text{echo}$ **and** $color_w = color_u$.

- **L3**(u): $mode_u = \text{echo} \Longrightarrow \forall w \in \mathsf{Children}_u$,

  - $color_u = color_{uw} = color_w$, and

  - $mode_{uw} = mode_w = \text{echo}$.

- $\mathbf{L}(u) \triangleq \mathbf{L1}(u) \wedge \mathbf{L2}(u) \wedge \mathbf{L3}(u)$.

Definition 7.10 (Well-coloredness) A tree T_r is *well-colored* in state s if it satisfies the following conditions:

1. All nodes $u \in T_r$ satisfy $\mathbf{L}(u)$ in s.

2. (Predicate $\mathbf{L}'$) At most two colors are contained in T_r, i.e.,

$$|\bigcup_{u \in T_r} s.color_u| \leq 2$$

Definition 7.11 $(\mathcal{GT}_r)$

$$\mathcal{GT}_r \triangleq \{s \in \mathcal{C}_{\overline{S}F}^{=} \mid T_r \text{ is well-colored}\}$$

80

The following Lemma shows that once a tree is well-colored, it stays well-colored:

Lemma 7.12 $\mathcal{GT}_r \implies \mathcal{GT}_r\boxdot$.

Proof. Let (s, a, s') be a step such that $s \in \mathcal{GT}_r$. Note that the only variables that are referenced by the coloring predicates are $color_u$, $mode_u$, and for all $v \in \mathsf{Children}_u$, $color_{uv}$ and $mode_{uv}$. We consider each $a \in acts(\mathsf{RSST})$, in turn:

Case 1 $a = \mathrm{COPY}_{uv}$.

Since u can only copy a color from v, $\mathbf{L}'$ must be true in s'. If $v \neq parent_u$ and $v \notin \mathsf{Children}_u$, the coloring predicates remain unchanged. If $v = parent_u$, then $[\mathbf{D}]$ may be executed. If $s.color_u \neq s.color_v$, then $s'.color_u = s'.color_v$, and $s'.mode_u = s'.mode_v = $ broadcast. Also, because $\mathbf{L}'$ holds in s, for all $w \in \mathsf{Children}_u$, $s.color_w = s.color_u$, which implies $s'.color_w \neq s'.color_u$. Thus $\mathbf{L1}(u)$, $\mathbf{L2}(u)$, and $\mathbf{L3}(u)$ are true in s'. Further, $\mathbf{L2}(v)$ holds in s'. Since $s.mode_w = s'.mode_w = $ echo for any child w of u, w satisfies $\mathbf{L1}$, $\mathbf{L2}$ and $\mathbf{L3}$ in s'.

If $v \in \mathsf{Children}_u$, $\mathbf{L1}(u)$, $\mathbf{L2}(u)$ and $\mathbf{L3}(u)$ continue to hold in s'.

Case 2 $a = \mathrm{MAXIMIZE\text{-}PRIORITY}_u$.

Since $s \in \mathcal{C}_{SF}^{=}$, all variables in s' are identical to those in s.

Case 3 $a = \mathrm{DETECT\text{-}TREES}_u$.

If $[\mathbf{K}]$ is executed then $s'.mode_u = $ echo. $\mathbf{L1}$ and $\mathbf{L2}$ are trivially satisfied, and $\mathbf{L3}$ is satisfied in s' because of the conditions in $[\mathbf{J}]$ and the fact that $\mathbf{L2}$ was satisfied in s.

Case 4 $a = \mathrm{NEXT\text{-}COLOR}_u$.

If the test in NEXT-COLOR is true, $u \in \rho(s)$, and $\mathbf{L3}$ implies that all nodes $v \in \mathrm{TREE}(u)$ have the same color c in s. Hence $\mathbf{L}'$ is satisfied. The coloring predicates can be seen to hold.

Case 5 $a = \text{EXTEND-ID}_u$.

If the test is satisfied, then $s' \in \mathcal{C}^1$. ∎

Definition 7.13 ($\mathcal{C}_{\overline{W}C}^{\overline{=}}$)

$$\mathcal{C}_{\overline{W}C}^{\overline{=}} \triangleq \{ s \in \mathcal{C}_{\overline{S}F}^{\overline{=}} \mid s \in \mathcal{GT}_r \quad \forall r \in \rho(s) \}$$

Corollary 7.14 $\mathcal{C}_{\overline{W}C}^{\overline{=}} \Longrightarrow \mathcal{C}_{\overline{W}C}^{\overline{=}} \boxdot$.

Proof. This is a direct consequence of Lemma 7.12 and Definition 7.13. ∎

Once a state is in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$, the coloring algorithm can proceed "normally" over all trees in the forest. We show that starting from a state in $\mathcal{C}_{\overline{S}F}^{\overline{=}}$, unless a state in $\mathcal{C}^1$ is reached, within time $17\delta + 7$ each tree becomes well-colored, so within time $17\delta + 7$ a state in $\mathcal{C}_{\overline{W}C}^{\overline{=}}$ is reached. Thus we show that $\mathcal{C}_{\overline{S}F}^{\overline{=}} \overset{17\delta+7}{\Longrightarrow} \mathcal{GT}_r$ for all roots r, which implies that $\mathcal{C}_{\overline{S}F}^{\overline{=}} \overset{17\delta+7}{\Longrightarrow} \mathcal{C}_{\overline{W}C}^{\overline{=}}$ (this is shown in Lemma 7.65).

Definition 7.15 (Monocolored, bicolored intervals and trees; $\mathcal{MT}_r$) A tree T_r is *monocolored* in $s \in \mathcal{C}_{\overline{S}F}^{\overline{=}}$ if it contains only one color, i.e. $color_u = c$ for some color c and all $u \in T_r$. (We say that T_r is *monocolored with color c*.) The set $\mathcal{MT}_r$ is defined as the set of states in $\mathcal{C}_{\overline{S}F}^{\overline{=}}$ in which T_r is monocolored. Similarly, a root interval is monocolored if it contains only one color. T_r is *bicolored* if it contains two colors (cf. Definition 7.10). ∎

The statement $\mathcal{C}_{\overline{S}F}^{\overline{=}} \overset{17\delta+7}{\Longrightarrow} \mathcal{GT}_r$ is proved using two main results: $\mathcal{C}_{\overline{S}F}^{\overline{=}} \overset{4\delta+1}{\Longrightarrow} \mathcal{MT}_r$ (the "Monocoloring" Result) and $\mathcal{MT}_r \overset{13\delta+6}{\Longrightarrow} \mathcal{GT}_r$ (the "Blocking" Result).

7.2.1 The "Monocoloring" Result

In this section we establish the first of the two self-stabilization results, $\mathcal{C}_{\overline{S}F}^{\overline{=}} \overset{4\delta+1}{\Longrightarrow} \mathcal{MT}_r$. Thus, starting from a state in $\mathcal{C}_{\overline{S}F}^{\overline{=}}$ defining a stable forest, any execution α reaches a state in which tree T_r is monocolored, within time $4\delta + 1$.

An overview of the proof follows. A *coloring epoch of color c* for T_r is defined as a maximal execution fragment contained in α in which the root color $color_r$ remains fixed at c; $color_r$ changes from one epoch to the next. As will be apparent from the code for COPY, if a node notices that it has a color different from that of its parent, it copies its parent's color. A *root-color interval* for a branch in T_r is the maximal root interval in the branch that has the same color as the root. Since children copy their parents' color, in any coloring epoch the root-color interval for any branch can only increase. Thus, in the last state of a coloring epoch γ, the root-color intervals in a tree are of maximal length; the *scope* of γ is the depth upto which the root color has propagated in epoch γ. Thus in the last state of an epoch γ, all root intervals of length $\leq \text{SCOPE}(\gamma)$ are colored with the root color.

Consider any branch B in T_r of scope m in some coloring epoch γ of color c. When the root chooses a new color c' and sets its *mode* to broadcast, thus initiating the next coloring epoch γ', all its descendants of depth $\leq m$ are colored c. Because a root must echo before it can choose the next color, all descendants of depth $\leq m+1$ must be colored with c' in coloring epoch γ'. Thus each coloring epoch has a higher scope than its predecessor (provided that this is feasible, i.e., the scope of its predecessor was not $\text{HEIGHT}(T_r)$). If a coloring epoch of scope $\text{HEIGHT}(T_r)$ is reached, there must exist some state in that epoch in which T_r is monocolored.

A finer analysis, in Lemmas 7.37 − 7.39, shows that if a coloring epoch γ' is of duration Δ, its scope is at least $\lfloor \Delta \rfloor$ higher than that of its predecessor γ (if feasible). Based on this progress property, Lemma 7.40 shows that the scope of a coloring epoch beginning after time t in α must be at least $t/2$. Thus we conclude, in Lemma 7.41, that within time 3δ an epoch of scope $\geq \text{HEIGHT}(T_r)$ is reached, and therefore, in Lemma 7.42, that a monocolored state is reached in time $\leq 4\delta + 1$.

Definition 7.16 (Root-color interval) Let $s \in \mathcal{C}_{SF}^{=}$; let T_r be a tree, and let $B \in$ BRANCHES(T_r). The *root-color interval* of B, denoted *RC(B)*, is the maximal prefix $u_0 \ldots u_i$ of B having the same color as the root u_0, i.e., for which $color(u) = color(u_0)$ for every $u \in RC(B)$. ∎

Definition 7.17 (Root-color extent) Let $B \in \text{BRANCHES}(T_r)$. The root-color extent of B, written $\text{EXTENT}(B)$, is defined as:

$$\text{EXTENT}(B) \triangleq \begin{cases} 1. \ |RC(B)| \text{, if } RC(B) \neq B \text{ (i.e., } RC(B) \text{ is a proper prefix of B)}. \\ 2. \ \text{HEIGHT}(T_r), \text{ if } RC(B) = B. \end{cases}$$

∎

Thus the root-color extent of a branch is the length of the maximal prefix that has the same color as the root, unless the whole branch has the same color, in which case it is the height of the tree.

Definition 7.18 (Root-color domain) The root-color domain of tree T_r, written $\text{DOMAIN}(T_r)$,

$$\text{DOMAIN}(T_r) \triangleq \min_{B \in \text{BRANCHES}(T_r)} \text{EXTENT}(B).$$

∎

Claim 7.19 *Let* (s, a, s') *be a step in* $\mathcal{C}_{\bar{S}F}^=$. *For any root* $r \in \rho(s) \cap \rho(s')$, *if* $s.color_r \neq s'.color_r$, *then* $a = \text{NEXT-COLOR}_r$.

Proof. From the code, the only statements that can change the color of r are **[D]** of COPY and the actions NEXT-COLOR_r and EXTEND-ID_r. Since $parent_r = \text{nil}$, **[D]** of COPY is not executed. If $a = \text{EXTEND-ID}_r$ and $s.color_r \neq s'.color_r$, then $s.ID_r \prec s'.ID_r$, and so $s' \notin \mathcal{C}_{\bar{S}F}^=$. Hence the only possibility for a is NEXT-COLOR_r. ∎

Definition 7.20 (Coloring epochs) Let α be an execution fragment in $\mathcal{C}_{\bar{S}F}^=$. A *coloring epoch* for tree T_r is a maximal execution fragment γ contained in α such that $color_r$ remains constant in γ. Let $\text{COLOR}(\gamma)$ denote the *color of epoch* γ, i.e. $s.color_r$ for any $s \in \gamma$. ∎

Observation 7.21 From Claim 7.19, for any tree T_r, any execution α in $\mathcal{C}_{\bar{S}F}^=$ contains coloring epochs $\gamma_1, \gamma_2, \gamma_3, \ldots$ for T_r, such that $\alpha = \gamma_1 a \gamma_2 a \gamma_3 a \ldots$, where $a = \text{NEXT-COLOR}_r$.

Claim 7.22 *If γ_i and γ_{i+1} are successive coloring epochs in some execution α then*

- $\text{COLOR}(\gamma_i) = 0 \implies \text{COLOR}(\gamma_{i+1}) \neq 0$

- $\text{COLOR}(\gamma_i) \neq 0 \implies \text{COLOR}(\gamma_{i+1}) = 0$

Proof. Follows from the code for NEXT-COLOR, New-Color() and Reset-Color(). ∎

Definition 7.23 (Scope) Let γ be a coloring epoch for T_r. The *scope* of a coloring epoch γ for T_r is

$$\text{SCOPE}(\gamma) \triangleq \max_{s \in \gamma} s.\text{DOMAIN}(T_r)$$

The scope of γ for a *branch B* in T_r is defined similarly:

$$\text{SCOPE}_B(\gamma) \triangleq \max_{s \in \gamma} s.\text{EXTENT}(B)$$

∎

Lemma 7.24 *Let $u \notin \rho(s)$ and let α be an execution fragment in $\mathcal{C}_{SF}^{\bar{=}}$ starting with s. In any step (s', a, s'') in α such that $s''.color_u \neq s'.color_u$, $s''.color_u = s'.color_{parent_u}$, $s''.mode_u = \text{broadcast}$, and for all $v \in \text{Children}_u$, $s''.color_{uv} = \text{undefined}$.*

Proof. From the code, a must be COPY_u, and [**D**] must be executed. ∎

Lemma 7.25 *In any step (s, a, s') in some coloring epoch γ for T_r, $s.color_u = s'.color_u$ for any $u \in s.RC(B)$, where $B \in \text{BRANCHES}(T_r)$.*

Proof. By induction on the depth of u in T_r. Let $s.RC(B) = u_1 \ldots u_i$. Since $s, s' \in \gamma$, $s.color_{u_1} = s'.color_{u_1}$. Suppose $s.color_{u_k} = s'.color_{u_k}$ for some $u_k \in u_1 \ldots u_{i-1}$. Since $u_{k+1} \in RC(B)$, $s.color_{u_{k+1}} = s.color_{u_k}$. If $s'.color_{u_{k+1}} \neq s.color_{u_{k+1}}$, then by Lemma 7.24 $s'.color_{u_{k+1}} = s.color_{u_k}$, which is a contradiction since $s.color_{u_{k+1}} = s.color_{u_k}$. ∎

Corollary 7.26 *Let γ be a coloring epoch. If (s, a, s') is a step in γ, for any branch $B \in$* BRANCHES(T_r), $s.RC(B)$ *is a prefix of $s'.RC(B)$.*

Proof. Let $B \in$ BRANCHES(T_r). From Lemma 7.25, for every $u \in RC(B)$, $s'.color_u = s.color_u = s.color_r$. Hence the Corollary follows. ∎

Corollary 7.27 *In any coloring epoch γ,* DOMAIN(T_r) *cannot decrease.*

Proof. Immediate from Definition 7.17 (EXTENT), Definition 7.18 (DOMAIN), and Corollary 7.26. ∎

Lemma 7.28 *Let α be an execution fragment contained in some coloring epoch for T_r. Let $t = (lstate(\alpha).now - fstate(\alpha).now)$. Then $lstate(\alpha).$DOMAIN$(T_r) \geq \min((fstate(\alpha).$DOMAIN$(T_r) + \lfloor t \rfloor),$ HEIGHT$(T_r))$.*

Proof. We show that for any $B \in$ BRANCHES(T_r), $lstate(\alpha).$EXTENT$(B) \geq \min(fstate(\alpha).$EXTENT$(B) + \lfloor t \rfloor,$ HEIGHT$(T_r))$. The Lemma then follows from the definition of DOMAIN(T_r) (Definition 7.18).

Consider any branch $B = u_1 \ldots u_l$ in T_r, and an execution fragment $\alpha = s_0 a_1 s_1 a_2 \ldots a_k s_k$ contained in some coloring epoch γ of color c for T_r. Let $t = s_k.now - s_0.now$, and let $t' = \lfloor t \rfloor$. Let $R = s_0.RC(B) = u_1 \ldots u_i$.

We show that if $l \geq i + t'$, i.e., the length of branch B is at least $i + t'$, then $u_1 \ldots u_{i+t'}$ is a prefix of $s_k.RC(B)$. Otherwise, $s_k.RC(B) = B$.

If $t' = 0$, then Corollary 7.26 implies that $u_1 \ldots u_i$ is a prefix of $s_k.RC(B)$.

If $t' \geq 1$, and if $l \geq i + 1$, then there must exist a step $(s, \text{COPY}_{u_{i+1}u_i}, s')$ in α such that $s'.now \leq s_0.now + 1$. Corollary 7.26 implies that $s.color_{u_i} = c$; hence $s'.color_{u_{i+1}} = c$, and $u_1 \ldots u_{i+1}$ is a prefix of $s'.RC(B)$ and $s_k.RC(B)$.

86

If $t'' \geq 2$ and $l \geq i + 2$, then there must exist a step $(s'', \text{COPY}_{u_{i+2}u_{i+1}}, s''')$ in α such that $s'''.now \leq s_0.now + 2$ and s'' follows s' in α. Again, Corollary 7.26 implies that $s''.color_{u_{i+1}} = c$; therefore $u_1 \ldots u_{i+2}$ is a prefix of $s''.RC(B)$.

The Lemma follows by proceeding inductively as above. ∎

Lemma 7.29 *Let $u, v \in T_r$, and let $parent_u = v$. Let $(s_0, \text{COPY}_{uv}, s_1)$ be a step in which $s_0.color_u \neq s_1.color_u$, and let $\alpha = s_0 a_1 s_1 a_2 \ldots$ be an execution fragment starting with this step, contained in some coloring epoch for T_r. Let w be a child of u. If there exists $s_i \in \alpha$ ($i \neq 0$) such that $s_i.mode_u = $ echo, then there exists s' between s_0 and s_i such that $s'.color_u = s'.color_w$.*

Proof. From the code, statement **[D]** in COPY_{uv} must have been executed in the first step, so from the code for Reset-Color(), $s_1.mode_u = $ broadcast, and $s_1.color_{uw} = $ undefined.

Since $s_i.mode_u = $ echo, there must exist a step (s', a, s'') between s_1 and s_i such that $s'.mode_u = $ broadcast and $s''.mode_u = $ echo. From the code, the only possibility for a is DETECT-TREES$_u$. From statement **[J]**, it follows that $s'.mode_{uw} = $ echo and $s'.color_{uw} = s'.color_u$. Since $s'.color_w = s'.color_{uw}$, $s'.color_u = s'.color_w$. ∎

Let *var* be one of the state components for a node (e.g. *mode*, *color*), and let *value* be one of the corresponding values that can be assumed by the state components (e.g. "broadcast," for the *mode* component). Henceforth, to ease the notation, the expression $var(u_1 u_2 \ldots u_k) = value$ will be used to denote the relation $var_{u_1} = var_{u_2} = \ldots = var_{u_k} = value$.

Definition 7.30 (Broadcast and echo intervals) Let $R = u_1 u_2 \ldots u_k$ be a root interval. Then,

- R is a *broadcast interval* if $mode(u_1 u_2 \ldots u_k) = $ broadcast, and $\mathbf{L}(u)$ is true for all u in R. (Note that the conditions of $\mathbf{L}$ imply that for such an interval, $color(u_1 u_2 \ldots u_k) = $ some color c, and for each u_j in $u_1 u_2 \ldots u_{k-1}$, $\neg(mode_{u_j u_{j+1}} = $ echo **and** $color_{u_j u_{j+1}} = c)$.) A *broadcast interval of color c* is a broadcast interval in which every node has color c ($color_u = c$).

- R is an *echo interval* if $mode(u_1u_2\ldots u_k) = $ echo, and $\mathbf{L}(u)$ is true for all u in R. (Note that the conditions of $\mathbf{L}$ imply that for such an interval, $color(u_1u_2\ldots u_k) = $ some color c, and for each u_j in $u_1u_2\ldots u_k$, ($mode_{u_ju_{j+1}} = $ echo **and** $color_{u_ju_{j+1}} = c$).) An *echo interval of color c* is an echo interval in which every node has color c. ∎

Lemma 7.31 *Let $R = u_1u_2\ldots u_k$ be a a broadcast interval of color c in s, and let α be an execution fragment in $\mathcal{C}_{SF}^{=}$ starting with s. If there exists s' in α such that $s'.color_u \neq s.color_u$ for some $u \in R$, then there exists s'' before s' in α such that R is an echo interval of color c in α.*

Proof. The coloring epoch containing s' must be different from that containing s. A new coloring epoch for R can only begin after a state s_1 in which $s_1.mode_{u_1} = $ echo. From the code for DETECT-TREES, s_1 must follow a state s_2 such that $s_2.mode_{u_1u_2} = $ echo and $s_2.color_{u_1u_2} = c$. Also, $s_2.mode_{u_2} = $ echo and $s_2.color_{u_2} = c$. Thus u_1 satisfies $\mathbf{L}(u)$. Proceeding inductively, s_2 must follow some state s_k in which $s_k.mode_{u_{k-1}u_k} = s_k.mode_{u_k} = $ echo, and $s_k.color_{u_{k-1}u_k} = s_k.color_{u_k} = c$. Hence R is an echo interval of color c in s_1. ∎

Lemma 7.32 *Let γ be a coloring epoch for T_r, and let s be a state in γ such that in a branch $B = u_1u_2\ldots u_k$ of T_r, there exist i,j such that $u_1\ldots u_i$ is a broadcast interval of color c, and $color(u_{i+1}\ldots u_j) = c' \neq c$. (Such an interval $u_1\ldots u_j$ is called* properly bicolored.*) Then,*

- *There exists s' following s in γ such that $u_1\ldots u_j$ is a broadcast interval of color c, and (therefore)*

- $\mathrm{SCOPE}_B(\gamma) \geq j$.

Proof. $u_1\ldots u_i$ is a broadcast interval of color c in state s. In any execution fragment α beginning with s, a new coloring epoch γ' can only begin after a state s_1 such that $s_1.mode_{u1} = $ echo (from the code for NEXT-COLOR). But since $s.mode_{u1} = $ broadcast and $\mathbf{L2}(u_1)$ holds in s, s_1 must follow some state s_2 in which $color_{u_2} = c$ and $mode_{u_2} = $ echo. Continuing

inductively, s_1 must follow some state s_{i+1} in which $color_{u_{i+1}} = c$ and $mode_{u_{i+1}} = $ echo. But s_{i+1} must follow some step $(s'_{i+1}, \mathrm{COPY}_{u_{i+1}u_i}, s''_{i+1})$ in which u_{i+1} "copies" color c from u_i; $u_1 \ldots u_{i+1}$ is a broadcast interval of color c in s''_{i+1}. Proceeding inductively, there must exist s' in which $u_1 \ldots u_j$ is a broadcast interval of color c. ∎

Claim 7.33 *Any prefix of a monocolored root interval is monocolored, and a prefix of a properly bicolored interval is monocolored or properly bicolored.*

Proof. Follows from the definitions. ∎

Claim 7.34 *Let $R = u_1 u_2 \ldots u_k$ be a root interval in T_r. Let γ be a coloring epoch for T_r, and let $s \in \gamma$. Then,*

1. *If R is monocolored in s, it is monocolored for all s' following s in γ.*

2. *If R is properly bicolored (cf. Lemma 7.32) in s, it is monocolored or properly bicolored for all s' following s in γ.*

Proof. Follows from Lemma 7.24 and Corollary 7.26. ∎

Corollary 7.35 *Let $\gamma_1 a \gamma_2$ be an execution fragment in $\mathcal{C}^{\bar{=}}_{SF}$ such that γ_1 and γ_2 are coloring epochs for T_r of colors c_1 and c_2 respectively. Let $\mathrm{SCOPE}(\gamma_1) = m$. Then for any root interval $R = u_1 \ldots u_m$ in T_r of length m, there exists $s \in \gamma_2$ such that $u_1 \ldots u_m$ is a broadcast interval of color c_2.*

Proof. Note that in $fstate(\gamma_2)$, $u_1 \ldots u_m$ is properly bicolored. The Corollary then follows from Lemma 7.32. ∎

Lemma 7.36 *Let $\gamma_1 a \gamma_2 a \gamma_3$ be an execution fragment in $\mathcal{C}^{\bar{=}}_{SF}$ such that γ_1, γ_2 and γ_3 are coloring epochs for T_r. Then $\mathrm{SCOPE}(\gamma_2) \geq \min(\mathrm{SCOPE}(\gamma_1) + 1, \mathrm{HEIGHT}(T_r))$.*

Proof. Let γ_1 and γ_2 be of colors c_1 and c_2 respectively. Let $\textsc{Scope}(\gamma_1) = m$; note that $\textsc{Height}(T_r) \geq m$. Let $B = u_1 \ldots u_k$ be a branch in T_r of height $\geq m$, and let $R = u_1 \ldots u_m$ be a prefix of B.

From Corollary 7.35, there exists a $s \in \gamma_2$ such that R is a broadcast interval of color c_2. If $k = m$, then $\textsc{Scope}_B(\gamma_2) = \textsc{Height}(T_r)$. If $k > m$, then $s.color_{u_{m+1}} = c_1$ or c_2. If $s.color_{u_{m+1}} = c_1$, then $u_1 \ldots u_{m+1}$ is a properly bicolored interval, so by Lemma 7.32 $\textsc{Scope}_B(\gamma_2) \geq m + 1$. If $s.color_{u_{m+1}} = c_2$, then $\textsc{Scope}_B(\gamma_2) \geq m + 1$ by definition. ■

Lemma 7.37 *Let $\gamma_1 a \gamma_2$ be an execution fragment in $\mathcal{C}^{=}_{\overline{S}F}$ such that γ_1 and γ_2 are coloring epochs for T_r, and let $\textsc{Scope}(\gamma_1) = m$.*

For any integer i, if $(lstate(\gamma_2).now - fstate(\gamma_2).now) \geq i$, then for any root interval $R = u_1 \ldots u_k$ of length $\leq (m + i)$, there exists a state $s \in \gamma_2$ such that $s.now \leq fstate(\gamma_2).now + i$, and R is either monocolored or properly bicolored in s.

Proof. By induction on i.

Base $(i = 0)$: Clearly, in $fstate(\gamma_2)$, any interval $u_1 \ldots u_k$ of length $\leq m$ is monocolored if $k = 1$, and is properly bicolored if $k > 1$.

Now suppose the Lemma holds for i. We need to show that it must hold for $i + 1$.

Consider any root interval $R = u_1 \ldots u_{(m+i+1)}$. Since the Lemma holds for i, there exists a state $s \in \gamma_2$ such that $s.now \leq fstate(\gamma_2).now + i$, and $u_1 \ldots u_{m+i}$ is either monocolored or properly bicolored in s. There must exist a step $(s_1, \text{COPY}_{u_{(m+i+1)} u_{(m+i)}}, s_2)$ in γ_2, such that s_1 follows s, and $(s_1.now \leq s.now + 1)$. Thus $s_2.now \leq (fstate(\gamma_2).now + i + 1)$. Consider the two cases:

Case 1 $u_1 \ldots u_{m+i}$ is monocolored in s.

Then, by Claim 7.34, $u_1 \ldots u_{m+i}$ must be monocolored in s_1, and therefore it must be monocolored in s_2. Hence the Lemma follows.

Case 2 $u_1 \ldots u_{m+i}$ is properly bicolored in s.

Then by Claim 7.34, $u_1 \ldots u_{m+i}$ is either monocolored or properly bicolored in s_1. If $u_1 \ldots u_{m+i}$ is monocolored in s_1, $u_1 \ldots u_{(m+i+1)}$ must be monocolored in s_2. If $u_1 \ldots u_{m+i}$ is properly bicolored in s_1, $u_1 \ldots u_{(m+i+1)}$ must be properly bicolored in s_2. Hence the Lemma follows.

$\blacksquare$

Corollary 7.38 *Let $\gamma_1 a \gamma_2$ be an execution fragment in $\mathcal{C}^=_{\overline{S}F}$ such that γ_1 and γ_2 are coloring epochs for T_r, and let $\textsc{Scope}(\gamma_1) = m$.*

For any integer i, if $(lstate(\gamma_2).now - fstate(\gamma_2).now) \geq i$, there exists a state $s \in \gamma_2$ such that $s.now \leq fstate(\gamma_2).now + i$, such that every root interval of length $\leq (m + i)$ is either monocolored or properly bicolored in s.

Proof. Follows from Lemma 7.37 and Claim 7.34. $\blacksquare$

Lemma 7.39 *Let $\gamma_1 a \gamma_2$ be an execution fragment in $\mathcal{C}^=_{\overline{S}F}$ such that γ_1 and γ_2 are coloring epochs for T_r, and let $\textsc{Scope}(\gamma_1) = m$. Let $\Delta = (lstate(\gamma_2).now - fstate(\gamma_2).now)$. Then*

$$\textsc{Scope}(\gamma_2) \geq \min(\ (\textsc{Scope}(\gamma_1) + \lfloor \Delta \rfloor)\ ,\ \textsc{Height}(T_r)\ ).$$

Proof. Let $\Delta' = \lfloor \Delta \rfloor$. From Lemma 7.38, there exists a state s in γ_2 such that $(s.now - fstate(\gamma_2).now) \leq \Delta'$, and every root interval of length $\leq m + \Delta'$ is either monocolored or properly bicolored in s. Hence by Lemma 7.32, $\textsc{Scope}_B(\gamma_2) \geq \min(m + \Delta', \textsc{Height}(T_r))$ for every branch B. Hence $\textsc{Scope}(\gamma_2) \geq \min(m + \Delta', \textsc{Height}(T_r))$. $\blacksquare$

Lemma 7.40 *Let $\alpha = \gamma_1 a \gamma_2 a \gamma_3 \ldots$ be an execution in $\mathcal{C}^=_{\overline{S}F}$, where γ_1, γ_2, $\gamma_3, \ldots$ are coloring epochs for tree T_r. Then for any coloring epoch γ,*

$$\textsc{Scope}(\gamma) \geq \min(\ fstate(\gamma).now/2\ ,\ \textsc{Height}(T_r)\ )$$

91

Proof. By induction on γ.

Clearly, $\text{SCOPE}(\gamma_1) \geq 0$.

Now suppose the Lemma holds for γ_i, i.e. $\text{SCOPE}(\gamma_i) \geq \min(\,fstate(\gamma_i).now/2\,,\,\text{HEIGHT}(T_r)\,)$. If $\text{SCOPE}(\gamma_i) = \text{HEIGHT}(T_r)$, then Lemma 7.36 implies $\text{SCOPE}(\gamma_{i+1}) = \text{HEIGHT}(T_r)$, which satisfies the Lemma. If $\text{SCOPE}(\gamma_i) < \text{HEIGHT}(T_r)$, then by the inductive hypothesis, $\text{SCOPE}(\gamma_i) \geq fstate(\gamma_i).now/2$. We show that $\text{SCOPE}(\gamma_{i+1}) > fstate(\gamma_{i+1}).now/2$, which would satisfy the Lemma. Consider the two cases:

Case 1 $(fstate(\gamma_{i+1}).now - fstate(\gamma_i).now) < 1$.

Then Lemma 7.36 yields

$$
\begin{aligned}
\text{SCOPE}(\gamma_{i+1}) \quad &\geq \quad \text{SCOPE}(\gamma_i) + 1 \\
&\geq \quad fstate(\gamma_i).now/2 \; + 1 \qquad \text{(by the inductive hyp.)} \\
&= \quad (fstate(\gamma_i).now + 2)/2 \\
&\geq \quad fstate(\gamma_{i+1}).now/2
\end{aligned}
$$

Case 2 $(fstate(\gamma_{i+1}).now - fstate(\gamma_i).now) \geq 1$.

Then by Lemma 7.39,

$$
\begin{aligned}
\text{SCOPE}(\gamma_{i+1}) \quad &\geq \quad \text{SCOPE}(\gamma_i) + \lfloor fstate(\gamma_{i+1}).now - fstate(\gamma_i).now \rfloor \\
&\geq \quad fstate(\gamma_i).now/2 \; + \; \lfloor fstate(\gamma_{i+1}).now - fstate(\gamma_i).now \rfloor \\
&\qquad \text{(by the inductive hypothesis)} \\
&> \quad fstate(\gamma_i).now/2 \; + \; (fstate(\gamma_{i+1}).now - fstate(\gamma_i).now)/2 \\
&\qquad \text{(since } x \geq 1 \text{ implies } \lfloor x \rfloor > x/2) \\
&= \quad fstate(\gamma_{i+1}).now/2
\end{aligned}
$$

$\blacksquare$

Lemma 7.41 *Let $\alpha = \gamma_1 a \gamma_2 a \gamma_3 \ldots$ be an execution fragment in $\mathcal{C}^{=}_{SF}$, where γ_1, γ_2, $\gamma_3,\ldots$ are coloring epochs for T_r. There exists an epoch γ_i in α such that $fstate(\gamma_i).now \leq 3\text{HEIGHT}(T_r)$, and $\text{SCOPE}(\gamma_i) = \text{HEIGHT}(T_r)$.*

Proof. If there exists an epoch γ_i in α such that $3\text{HEIGHT}(T_r) \geq fstate(\gamma_i).now \geq 2\text{HEIGHT}(T_r)$, then by Lemma 7.40 $\text{SCOPE}(\gamma_i) = \text{HEIGHT}(T_r)$. If there is no such epoch γ_i, then there must exist an epoch γ_i such that $fstate(\gamma_i).now < 2\text{HEIGHT}(T_r)$ and $lstate(\gamma_i).now > 3\text{HEIGHT}(T_r)$. Since $\lfloor lstate(\gamma_i).now - fstate(\gamma_i).now \rfloor \geq \text{HEIGHT}(T_r)$, Lemma 7.39 implies that $\text{SCOPE}(\gamma_i) = \text{HEIGHT}(T_r)$. ∎

Lemma 7.42 $\mathcal{C}_{\bar{S}F}^{=} \overset{4\delta+1}{\Longrightarrow} \mathcal{MT}_r, \ \forall r \in \rho$.

Proof. Let $s \in \mathcal{C}_{\bar{S}F}^{=}$. Let α be any execution fragment in $\mathcal{C}_{\bar{S}F}^{=}$ beginning with s, for which $(lstate(\alpha).now - fstate(\alpha).now) \geq 4\delta + 1$. Let $\alpha = \gamma_1 a \gamma_2 a \gamma_3 \ldots$, where $\gamma_1, \gamma_2, \gamma_3, \ldots$ are coloring epochs. By Lemma 7.41, there exists an epoch γ_i such that $fstate(\gamma_i).now \leq 3\text{HEIGHT}(T_r)$ and $\text{SCOPE}(\gamma_i) = \text{HEIGHT}(T_r)$.

If $lstate(\gamma_i).now \leq 4\text{HEIGHT}(T_r)$, then since $\text{SCOPE}(\gamma_i) = \text{HEIGHT}(T_r)$, there exists a state $s' = lstate(\gamma_i)$ such that $s'.now \leq 4\text{HEIGHT}(T_r) + 1$ and $s' \in \mathcal{MT}_r$.

If $lstate(\gamma_i).now > 4\text{HEIGHT}(T_r)$, then since $fstate(\gamma_i).now \leq 3\text{HEIGHT}(T_r)$, Lemma 7.28 implies that for any state s' in γ_i such that $4\text{HEIGHT}(T_r) < s'.now \leq (4\text{HEIGHT}(T_r) + 1)$, $s'.\text{DOMAIN}(T_r) = \text{HEIGHT}(T_r)$, which implies that $s' \in \mathcal{MT}_r$.

Since $\text{HEIGHT}(T_r) \leq \delta$, the Lemma follows. ∎

7.2.2 The "Blocking" Result

In this section we establish the second of the two self-stabilization results, $\mathcal{MT}_r \overset{13\delta+6}{\Longrightarrow} \mathcal{GT}_r$. Thus, starting from a state in $\mathcal{C}_{\bar{S}F}^{=}$ in which T_r is monocolored, any execution reaches a state in which tree T_r is well-colored, within time $13\delta + 6$.

If a tree T_r is monocolored with some color c in some state s, it stays monocolored until the root chooses a new color c'. When the new color c' is propagated to all nodes in the tree (as it must be, from Lemma 7.36), the tree becomes well-colored, since in the process of copying a new color from its parent a node resets its own coloring variables (through Reset-Color$_u$). We show, in Lemma 7.62, that within $12\delta + 6$ time the root must choose a new color.

93

In order to choose a new color, the root must first set its *mode* to echo (from the code), which requires that all its children echo. A node u could be prevented from echoing because it may be *blocked* by its neighbors—if its color is non-zero, it needs to notice a non-zero color at each of its neighbors (i.e., *nbr-color*$_{uv} \neq$ undefined), and it needs to notice that all neighbors have observed its color (*self-color*$_{uv} = color_u$). Theorem 7.60 shows that a node can be blocked for at most $10\delta + 5$ time, which implies that an "echo wave" must reach the root and cause it to choose a new color within $12\delta + 6$ time.

Definition 7.43 (Waiting) A node u *waits* in state $s \in \mathcal{C}_{SF}^{\bar{=}}$ if it is in a broadcast interval (cf. Definition 7.30). It *waits with color* c if it is waiting in s and $s.color_u = c$. ∎

Definition 7.44 (Waiting epoch) Let α be an execution fragment in $\mathcal{C}_{SF}^{\bar{=}}$. A *waiting epoch* ω for u is a maximal fragment contained in α such that u waits in each state of ω and $color_u$ remains constant in ω. A *waiting epoch of color* c is a waiting epoch in which u waits with color c. ∎

Definition 7.45 (Blocking, enabling) Let u be waiting in s with color $c \neq 0$, and let $v \in Nbrs(u)$. Then,

- u is *blocked by* v *on self-color* in s if $s.self\text{-}color_{uv} \neq color_u$. Otherwise, u is *enabled* by v on *self-color*.

- u is *blocked by* v *on nbr-color* in s if $s.nbr\text{-}color_{uv} =$ undefined. Otherwise, u is *enabled* by v on *nbr-color*.

- u is *blocked by* v in s if it is blocked by v on *self-color* or *nbr-color*.

- u is *enabled by* v in s if it is enabled by v on both *self-color* and *nbr-color*. ∎

Definition 7.46 (Recoloring) A node u is *recolored* in a step (s, a, s') if $s.color_u \neq s'.color_u$.
∎

Lemma 7.47 *Let $r \in \rho(s)$, and let α be an execution fragment in $\mathcal{C}_{SF}^=$ starting with s. If $s.mode_r =$ broadcast, and if there exists $s' \in \alpha$ such that $s'color_r \neq s.color_r$, then there exists a state s'' preceding s' in α such that $s''.color_r = s.color_r$ and $s''.mode_r =$ echo.*

Proof. Let (s_1, a, s_2) be the first step in α such that $s_1.color_r \neq s_2.color_r$; there must exist such a step between s and s' in α. From the code, a can only be NEXT-COLOR$_r$. Since $s_1.color_r = s.color_r$, and since $s_1.mode_r =$ echo from the condition in NEXT-COLOR$_r$, the Lemma follows. ∎

Claim 7.48 *In any step (s, a, s') in $\mathcal{C}_{SF}^=$ such that $s.mode_u =$ broadcast and $s'.mode_u =$ echo, u is enabled by all $v \in Nbrs(u)$ in s.*

Proof. Follows since a can only be DETECT-TREES$_u$ and the conditions in [**J**] must be satisfied. ∎

Lemma 7.49 *Let u be waiting in s, and let α be any execution fragment in $\mathcal{C}_{SF}^=$ starting with s. If there exists a step (s', a, s'') in α in which u is recolored, then there must exist a state s_1 between s and s'' in α such that u is enabled by all $v \in Nbrs(u)$ in s_1.*

Proof. Since u is waiting in s, there exists a broadcast interval $R = u_1 u_2 \ldots u$ in s. From Lemma 7.31, there exists s_2 between s and s'' such that R is an echo interval in s_2. Since $s_2.mode_u =$ echo, the Lemma follows from Claim 7.48. ∎

Lemma 7.50 *Let ω be a waiting epoch for u of color c. In any state $s \in \omega$ such that $(s.now > fstate(\omega).now + 2)$, u is enabled by all $v \in Nbrs(u)$ on self-color.*

Proof. Let ω be a waiting epoch of color c. For any $v \in Nbrs(u)$, there must exist a step $(s_1, COPY_{vu}, s_2)$ in ω such that $s_1.now = s_2.now \leq fstate(\omega).now + 1$. Since $s_1.color_u = c$, $s_2.color_{vu} = c$. There must exist another step $(s_3, COPY_{uv}, s_4)$ following s_2 in ω such that $s_3.now = s_4.now \leq s_2.now + 1$. Since $s_3.color_{vu} = c$, $s_4.self\text{-}color_{uv} = c$. Hence u is enabled by v on self-color in s_4. Further, for all states s following s_4 in ω u must remain enabled by v on self-color. Hence the Lemma follows. ∎

95

Lemma 7.51 *Let α be an execution fragment of duration > 1 contained in a waiting epoch for u. If u is blocked by v on nbr-color in lstate(α), then lstate(α).color$_{uv}$ = 0.*

Proof. Consider the last step (s, a, s') in α such that $a = \text{COPY}_{uv}$. Since *lstate*(α).*nbr-color$_{uv}$* = undefined, and COPY_{uv} is the only action that can change *nbr-color$_{uv}$* between s' and *lstate*(α), it follows that $s'.nbr\text{-}color_{uv}$ = undefined. Since statement [**C**] must have been executed in the COPY_{uv} step, $s'.color_v = 0$, and therefore $s'.color_{uv} = 0 = lstate(\alpha).color_{uv}$. ∎

Lemma 7.52 *Let ω be a waiting epoch for u. If u is enabled by v on nbr-color in some $s \in \omega$, u is enabled by v on nbr-color for all s' following s in ω.*

Proof. From the code, if $s.nbr\text{-}color_{uv} \neq$ undefined, the only code that can change $s.nbr\text{-}color_{uv}$ to undefined is the call to Reset-Color(), which can be made either through [**D**] of COPY_{uv} or through NEXT-COLOR_u or EXTEND-ID_u. Since ω is a waiting epoch for u, none of these possibilities is feasible. ∎

Lemma 7.53 *Let s be a state in $\mathcal{C}^{=}_{SF}$ in which u is blocked by v on nbr-color, $s.color_{uv} = 0$ or undefined, and v is blocked by u on self-color. Let α be any execution fragment in $\mathcal{C}^{=}_{SF}$ beginning with s. Then if there exists $s' \in \alpha$ such that v is enabled by u on self-color, there exists s'' before s' in α such that u is enabled by v on nbr-color.*

Proof. $s.color_{uv} = 0$ or undefined, $s.self\text{-}color_{vu} \neq s.color_v$, and $s'.self\text{-}color_{vu} = s'.color_v$. From Lemma 7.49, it is possible to choose an s' in α satisfying the given conditions such that $s'.color_v = s.color_v$. Since $s.self\text{-}color_{vu} \neq s.color_v$ and $s'.self\text{-}color_{vu} = s.color_v$, there must exist a step $(s_1, \text{COPY}_{vu}, s_2)$ between s and s' in α such that $s_1.self\text{-}color_{vu} \neq s.color_v$ and $s_2.self\text{-}color_{vu} = s.color_v$. Hence $s_1.color_{uv} = s.color_v \neq 0$. Since $s.color_{uv} = 0$ or undefined, and $s_1.color_{uv} > 0$, there must exist a step $(s_3, \text{COPY}_{uv}, s_4)$ between s and s_1 such that $s_3.color_{uv} = 0$ or undefined and $s_4.color_{uv} > 0$. Since $s_3.color_v \neq 0$, statement [**C**] in COPY_{uv} sets $s_4.nbr\text{-}color_{uv} = s_3.color_v \neq$ undefined. Thus u is enabled by v on nbr-color in s_4. ∎

Lemma 7.54 *Let s be a state in $\mathcal{C}_{\overline{S}F}^{=}$ in which u is blocked by v on nbr-color, $s.color_{uv} = 0$ or undefined, and v is blocked by u on self-color. Then in any execution fragment α in $\mathcal{C}_{\overline{S}F}^{=}$ beginning with s, there exists s' following s in α such that $s'.now \leq s.now + 1$ and u is enabled by v on nbr-color.*

Proof. There exist two exhaustive possibilities:

Case 1 There exists s' following s in α such that $s'.now \leq s.now + 1$ and v is enabled by u on *self-color*.

Then from Lemma 7.53, there exists s'' before s' in α such that u is enabled by v on *nbr-color*, and the Lemma follows.

Case 2 There exists no s' following s in α such that $s'.now \leq s.now + 1$ and v is enabled by u on *self-color*.

There must exist a step $(s_1, \text{COPY}_{uv}, s_2)$ in α such that $s_1.now \leq s.now + 1$. Since v is not enabled by u between s and s_1, $s_1.color_v = s.color_v > 0$. From statement [**C**] in COPY_{uv}, $s_2.nbr\text{-}color_{uv} = s_1.color_v > 0$; hence the Lemma follows. ∎

Lemma 7.55 *Let (s, a, s') be a step in $\mathcal{C}_{\overline{S}F}^{=}$ in which $s'.color_u \neq s.color_u$. Then u is blocked by all $v \in Nbrs(u)$ in s'.*

Proof. Follows since a must have called Reset-Color. ∎

Lemma 7.56 *Let T_r be monocolored with color 0 in s. In any execution fragment in $\mathcal{C}_{\overline{S}F}^{=}$ of duration $> 2\delta$ beginning with s, there exists a state s_1 in which $mode_r = $ echo.*

Proof. From the code in statement [**J**] in DETECT-TREES, nodes with color 0 do not "wait" for neighbors to enable them before echoing; a node u with color 0 echoes as soon as it notices that all its children are echoing. Thus the root must echo within time 2δ. ∎

Lemma 7.57 *Let* TREE(u) *be monocolored with color 0 in* $s \in \mathcal{C}_{\overline{S}F}^=$*. For any execution fragment* α *in* $\mathcal{C}_{\overline{S}F}^=$ *beginning with* s*, there exists* s' *following* s *in* α *such that* $s'.now \leq s.now + 4\delta$*, u waits in* s'*, and u is blocked by all neighbors* $v \in Nbrs(u)$ *in* s'*.*

Proof. By Lemma 7.56, within time 2δ in α a state is reached in which $mode_r = $ echo. Thus within time $2\delta + 1$, r must choose a new color (through NEXT-COLOR$_u$). Within δ additional time, u must be recolored with this new color. The Lemma follows from Lemma 7.55. ∎

Lemma 7.58 *Let* TREE(u) *be monocolored with a color* $\neq 0$ *in* s*, and let* α *be an execution fragment in* $\mathcal{C}_{\overline{S}F}^=$ *beginning with* s*. If there exists a state* s' *following* s *in* α *such that* $s'.now \leq s.now + 1$ *and* $s'.color_u = 0$*, then there exists a state* s'' *following* s *in* α *such that* $s'' \leq s + 1 + 2\delta$ *and* TREE(u) *is monocolored with color 0 in* s''*.*

Proof. Let $T_r = $ TREE(u). Since non-root nodes can only copy new colors from their parents, the coloring epoch γ' containing s' is different from the epoch γ containing s. Since T_r is monocolored in s, SCOPE$(\gamma) = $ HEIGHT(T_r). Hence from Lemma 7.36, SCOPE$(\gamma') = $ HEIGHT(T_r). Since γ' is of color 0, there exists a state s'' in γ' such that T_r is monocolored with color 0. Since $s'.color_u = 0$, and each child copies its parent's color within time 2, such a state s'' exists for which $s''.now < s'.now + 2\delta$. ∎

Lemma 7.59 *Let u be blocked by v on nbr-color in s, and let α be a fragment starting with s that is contained in some waiting epoch for u. If there exists an execution fragment α_1 in α such that $(lstate(\alpha).now - fstate(\alpha).now > 1)$ and $s'.color_v \neq 0$ for every $s' \in \alpha_1$, then u is enabled by v on nbr-color in $lstate(\alpha_1)$.*

Proof. There must exist a step $(s_1, \text{COPY}_{uv}, s_2)$ in α_1. Since $s_1.color_v > 0$, [**C**] in the code for COPY$_{uv}$ sets $s_2.nbr\text{-}color_{uv} = s_1.color_v > 0$, and so u is enabled by v on $nbr\text{-}color$ in s_2. The Lemma then follows from Lemma 7.52. ∎

Theorem 7.60 *Let α be an execution fragment in $\mathcal{C}_{SF}^{=}$, and let ω be a waiting epoch of duration $> (10\delta + 5)$ contained in α. In any $s' \in \omega$ such that $s'.now > fstate(\omega).now + (10\delta + 5)$, u is enabled by all $v \in Nbrs(u)$ on nbr-color.*

Proof. Let $s = fstate(\omega)$, and let u be blocked by some neighbor v on *nbr-color* in s. Let s_1 be a state in ω such that $(s.now + 1 < s_1.now \leq s.now + 2)$. If u is blocked by v on *nbr-color* in s_1, then by Lemma 7.51 $s_1.color_{uv} = 0$. By Lemma 7.42, there exists s_2 following s_1 in α such that $s_2.now \leq (s_1.now + 4\delta + 1)$ and TREE(v) is monocolored in s_2. If u is blocked by v on *nbr-color* in s_2, by Lemma 7.51 $s_2.color_{uv} = 0$. Note that $s_2.now \leq s.now + (4\delta + 3)$. Consider the two cases:

Case 1 TREE(v) is monocolored with color 0 in s_2.

By Lemma 7.57, there exists s_3 following s_2 in α such that $s_3.now \leq s_2.now + 4\delta$ and v is blocked by u in s_3. If u is blocked by v on *nbr-color* in s_3, Lemma 7.51 implies that $s_3.color_{uv} = 0$. Then by Lemma 7.54, there exists s_4 following s_3 in α such that $s_4.now \leq s_3.now + 1$ and u is enabled by v on *nbr-color*. Note that $(s_4.now \leq s.now + (4\delta + 3) + 4\delta + 1) = (s.now + 8\delta + 4)$.

Case 2 TREE(v) is monocolored with some color $\neq 0$ in s_2.

Then there must exist a step $(s_3, \text{COPY}_{uv}, s_4)$ such that s_3 follows s_2 in α and $s_3.now \leq s_2.now + 1$. If $s_3.color_v \neq 0$, u is enabled by v in s_4 on *nbr-color*. (Note that $s_4.now \leq s.now + (4\delta + 3) + 1 = s.now + 4\delta + 4$.) If $s_3.color_v = 0$, by Lemma 7.58 there exists a state s_4 following s_2 in α such that $(s_4.now \leq s_2.now + 1 + 2\delta)$ and TREE(v) is monocolored with color 0 in s_4. We now proceed as in Case 1 and conclude that there exists s_5 following s_4 in α such that $(s_5.now \leq s_4.now + (4\delta + 1))$ and u is enabled by v on *nbr-color* in s_5. Note that $s_5.now \leq (s_2.now + 6\delta + 2) \leq (s.now + 10\delta + 5)$.

By Lemma 7.52, u is enabled by v on *nbr-color* for all s' following s in ω such that $s'.now > (fstate(\omega).now + 10\delta + 5)$. ∎

Lemma 7.61 *Let α be an execution fragment in $\mathcal{C}_{SF}^{\equiv}$, and let ω be a waiting epoch of duration $> (10\delta + 5)$ contained in α. In any $s' \in \omega$ such that $s'.now > fstate(\omega).now + (10\delta + 5)$, $(s'.self\text{-}color_{uv} = color_u)$ **and** $(s'.nbr\text{-}color_{uv} \neq \text{undefined})$.*

Proof. Follows from Definition 7.45, Lemma 7.50, and Theorem 7.60. ∎

Lemma 7.62 *Let $s \in \mathcal{MT}_r$. In any execution fragment α in $\mathcal{C}_{SF}^{\equiv}$ of duration $\geq 12\delta + 6$ beginning with s, there exists a step $(s', \text{NEXT-COLOR}_r, s'')$ in α such that $(s'.now \leq s.now + 12\delta + 6)$, $s'.color_r \neq s''.color_r$, and $s' \in \mathcal{MT}_r$.*

Proof. This is a consequence of Lemma 7.61 and the fact that a node enabled by all its neighbors echoes at most 2 time units after all its children have echoed. ∎

Lemma 7.63 *Let (s, a, s') be a step in $\mathcal{C}_{SF}^{\equiv}$ such that $s \in \mathcal{MT}_r$ and $s'.color_r \neq s.color_r$. Then in any execution fragment α in $\mathcal{C}_{SF}^{\equiv}$ of duration $> \delta$ beginning with (s, a, s'), there exists s'' in α such that $s''.now \leq s.now + \delta$ and $s'' \in \mathcal{GT}_r$.*

Proof. Let $c' = s'.color_r$. Each branch $B \in \text{BRANCHES}(T_r)$ is properly bicolored in s', and thus by Lemma 7.32, for each branch B there exists a state s_B such that B is a broadcast interval of color c'. State s_B must be reached within time δ (since color c' can take upto δ time to propagate); any state following the latest such s_B in α must be in $\mathcal{GT}_r$. ∎

Lemma 7.64 $\mathcal{MT}_r \overset{13\delta+6}{\Longrightarrow} \mathcal{GT}_r$.

Proof. Follows from Lemmas 7.62 and 7.63. ∎

7.2.3 Self-stabilization of the Coloring Algorithm: Main Result

Lemma 7.65 $\mathcal{C}_{SF}^{\equiv} \overset{17\delta+7}{\Longrightarrow} \mathcal{C}_{WC}^{\equiv}$.

Proof. For all $r \in \Phi$, from Lemma 7.42 $\mathcal{C}_{\overline{S}F}^= \overset{4\delta+1}{\Longrightarrow} \mathcal{MT}_r$, and from Lemma 7.64, $\mathcal{MT}_r \overset{13\delta+6}{\Longrightarrow} \mathcal{GT}_r$. Hence for all r, $\mathcal{C}_{\overline{S}F}^= \overset{17\delta+7}{\Longrightarrow} \mathcal{GT}_r$. Since $\mathcal{GT}_r \Longrightarrow \mathcal{GT}_{r\square}$ by Lemma 7.12, the Lemma follows. ■

Lemma 7.66 (Main coloring self-stabilization result) $\mathcal{C}^= \overset{19\delta+8}{\Longrightarrow} \mathcal{C}_{\overline{W}C}^=$.

Proof. From Lemma 7.7, $\mathcal{C}^= \overset{2\delta+1}{\Longrightarrow} \mathcal{C}_{\overline{S}F}^=$. From Lemma 7.65, $\mathcal{C}_{\overline{S}F}^= \overset{17\delta+7}{\Longrightarrow} \mathcal{C}_{\overline{W}C}^=$. Thus the Lemma follows. ■

7.3   Tree Detection

From Lemma 7.66, starting from any state in $\mathcal{C}^=$, within time $19\delta+8$, unless a state in $\mathcal{C}^1$ is reached, a state in $\mathcal{C}_{\overline{W}C}^=$ is reached, which implies that all trees are well-colored. Thus the coloring algorithm can proceed "normally."

In this section we show that the coloring algorithm achieves its goal of detecting the existence of multiple trees with the same root ID, by showing that $\mathcal{C}_{\overline{W}C}^= \overset{58\delta+28}{\underset{2/9}{\longrightarrow}} \mathcal{C}^1$.

Definition 7.67 (Neighboring trees) *Trees T_r and $T_{r'}$ are said to be* neighbors *if there exist $u \in T_r$ and $v \in T_{r'}$ such that $v \in Nbrs(u)$.*

Let α be any execution starting with a state in $\mathcal{C}_{\overline{W}C}^=$, and let α' be the maximal prefix of α that is in $\mathcal{C}_{\overline{W}C}^=$, if α is finite, or α itself, if it is infinite. Let T and $\overline{T}$ be neighboring trees. From Observation 7.21, α' can be partitioned into *coloring epochs* γ_i for T and $\overline{\gamma}_i$ for $\overline{T}$ such that $\alpha' = \gamma_1 a \gamma_2 a \gamma_3 \ldots = \overline{\gamma}_1 a \overline{\gamma}_2 a \overline{\gamma}_3 \ldots$.

Definition 7.68 (γ_i notices $\overline{\gamma}_j$) Let T and $\overline{T}$ be neighboring trees. Let γ_i and $\overline{\gamma}_j$ be coloring epochs for T and $\overline{T}$ respectively, and let $\text{COLOR}(\gamma_i), \text{COLOR}(\overline{\gamma}_j) \neq 0$. Then, in execution α, γ_i *notices* $\overline{\gamma}_j$ if there exists a step $(s, \text{COPY}_{uv}, s')$ in α such that $u \in T$, $v \in \overline{T}$, $v \in Nbrs(u)$, and:

1. $s \in \gamma_i$, $s \in \overline{\gamma}_j$;

2. $s.color_u = \mathrm{COLOR}(\gamma_i)$, $s.color_v = \mathrm{COLOR}(\overline{\gamma}_j)$;

3. $s.mode_u = $ broadcast.

If these conditions hold, we also say that γ_i *notices* $\overline{\gamma}_j$ *in step* $(s, \mathrm{COPY}_{uv}, s')$. ∎

Definition 7.69 (γ_i confronts $\overline{\gamma}_j$) γ_i *confronts* $\overline{\gamma}_j$ if γ_i notices $\overline{\gamma}_j$ and $\mathrm{COLOR}(\gamma_i) \neq \mathrm{COLOR}(\overline{\gamma}_j)$.

Lemma 7.70 *Any coloring epoch γ for a tree T_r has duration $\leq 13\delta + 6$.*

Proof. If $fstate(\gamma) \notin \mathcal{MT}_r$, then because a new color propagates within one time unit from a parent to its child, there exists a state $s \in \gamma$ such that $s.now \leq fstate(\gamma).now + \delta$ and $s \in \mathcal{MT}_r$. From Lemma 7.62, there exists a step $(s', \mathrm{NEXT\text{-}COLOR}_r, s'')$ such that $s'.now \leq s.now + 12\delta + 6$ and $s'.color_r \neq s''.color_r$. Thus s'' begins a new coloring epoch. The Lemma follows from the fact that $s''.now \leq fstate(\gamma).now + 13\delta + 6$. ∎

Lemma 7.71 *Any coloring epoch γ of color 0 for a tree T_r has duration $\leq 3\delta + 2$.*

Proof. Similar to that of Lemma 7.70, with the exception that from statement **[J]**, a node colored 0 does not need to be enabled by its neighbors in order to echo. ∎

Lemma 7.72 *If γ_i confronts $\overline{\gamma}_j$, there exists s' following $fstate(\gamma_i)$ in α such that $s' \in \mathcal{C}^1$ and $s'.now < fstate(\gamma_i).now + (13\delta + 6)$.*

Proof. If γ_i confronts $\overline{\gamma}_j$, some node in T must set *other-trees* to true in γ within time $11\delta + 5$, since all nodes in T cannot remain broadcasting for more than time $11\delta + 5$. By time $13\delta + 5$, the root of T must set *other-trees* to true, and by time $13\delta + 6$, it must extend its ID by executing $\mathrm{EXTEND\text{-}ID}_u$, thus reaching a state in $\mathcal{C}^1$. ∎

Lemma 7.73 *There exists $i \leq 3$ such that γ_i notices $\overline{\gamma}_j$ for some j.*

Proof. Consider the following possibilities:

Case 1 $\mathrm{COLOR}(\gamma_1) = 0$.

Then by Claim 7.22 $\mathrm{COLOR}(\gamma_2) \neq 0$. There must then exist a step (s, a, s') in γ_2 such that $s.color_u = 0$, $s'.color_u = \mathrm{COLOR}(\gamma_2)$, $s'.mode_u = $ broadcast, and $s'.nbr\text{-}color_{uv} = $ undefined. Since u is blocked by v on *nbr-color* in s', there must exist another step $(s'', \mathrm{COPY}_{uv}, s''')$ following s' in γ_2 such that $s''.nbr\text{-}color_{uv} = $ undefined and $s'''.nbr\text{-}color_{uv} > 0$. s'' must then belong in some epoch $\overline{\gamma}_j$ for $\overline{T}$, such that $\mathrm{COLOR}(\overline{\gamma}_j) \neq 0$. From the definitions, γ_2 notices $\overline{\gamma}_j$.

Case 2 $\mathrm{COLOR}(\gamma_1) \neq 0$.

If there exists a state s' in γ_1 such that $s'.color_u = \mathrm{COLOR}(\gamma_1)$, $s'.mode_u = $ broadcast and $s'.nbr\text{-}color_{uv} = $ undefined, then by an argument similar to that in Case 1, γ_1 notices some $\overline{\gamma}_j$. If there exists no such s', we use the fact that $\mathrm{COLOR}(\gamma_2) = 0$ (Claim 7.22). Then, by reasoning identical to that in Case 1, γ_3 must notice some $\overline{\gamma}_j$. ∎

7.3.1 The "Order" Results

Claim 7.74 *Let $i < i'$. If γ_i notices $\overline{\gamma}_j$ and $\gamma_{i'}$ notices $\overline{\gamma}_{j'}$, then $j \leq j'$.*

Proof. Let γ_i notice $\overline{\gamma}_j$ in step (s_1, a, s_2) and $\gamma_{i'}$ notice $\overline{\gamma}_{j'}$ in step (s_3, a, s_4). Since γ_i precedes $\gamma_{i'}$ in α', s_2 precedes s_3. Since $s_1 \in \overline{\gamma}_j$ and $s_3 \in \overline{\gamma}_{j'}$, $\overline{\gamma}_j$ cannot follow $\overline{\gamma}_{j'}$ in α'; hence $j \leq j'$. ∎

Claim 7.75 *Let γ_i notice $\overline{\gamma}_j$ and $\overline{\gamma}_{j'}$ notice $\gamma_{i'}$. Then,*

1. $(j < j') \Longrightarrow (i \leq i')$.

2. $(j > j') \Longrightarrow (i \geq i')$.

Proof. Let γ_i notice $\overline{\gamma}_j$ in step (s_1, a, s_2) and let $\overline{\gamma}_{j'}$ notice $\gamma_{i'}$ in step (s_3, a, s_4).

If $(j < j')$, $\overline{\gamma}_j$ precedes $\overline{\gamma}_{j'}$, so s_2 precedes s_3. Hence γ_i must precede or coincide with $\gamma_{i'}$, and $i \leq i'$.

If $(j > j')$, $\overline{\gamma}_j$ follows $\overline{\gamma}_{j'}$, so s_4 precedes s_1. Thus γ_i cannot precede $\gamma_{i'}$, and $i \geq i'$. ∎

Lemma 7.76 *Let $i < i'$, and let γ_i notice $\overline{\gamma}_j$ and $\gamma_{i'}$ notice $\overline{\gamma}_{j'}$. For any coloring epoch $\overline{\gamma}_{j''}$ such that $j < j'' < j'$, if $\overline{\gamma}_{j''}$ notices some coloring epoch $\gamma_{i''}$, then $i \leq i'' \leq i'$.*

Proof. From Claim 7.75(1), $i \leq i''$, and from Claim 7.75(2), $i' \geq i''$. Hence $i \leq i'' \leq i'$. ∎

Lemma 7.77 *Let $\textsc{Color}(\gamma_i) = \textsc{Color}(\overline{\gamma}_j) \neq 0$, and let $\textsc{Color}(\gamma_i)$, $\textsc{Color}(\gamma_{i+2})$, and $\textsc{Color}(\overline{\gamma}_{j+2})$ all be different. If γ_i notices $\overline{\gamma}_j$, then either γ_{i+2} confronts ($\overline{\gamma}_j$ or $\overline{\gamma}_{j+2}$) or $\overline{\gamma}_{j+2}$ confronts (γ_i or γ_{i+2}).*

Proof. γ_{i+2} must notice $\overline{\gamma}_k$ for some k. From Claim 7.74, $k \geq j$. If $k = j$, γ_{i+2} confronts $\overline{\gamma}_j$. (Note that $k \neq j + 1$, since $\textsc{Color}(\overline{\gamma}_{j+1}) = 0$.) If $k = j + 2$, γ_{i+2} confronts $\overline{\gamma}_{j+2}$. Suppose $k > j + 2$. $\overline{\gamma}_{j+2}$ must then notice some $\gamma_{i'}$, and by Lemma 7.76, $i \leq i' \leq i + 2$. Since $\textsc{Color}(\gamma_{i+1}) = 0$, $i' \neq i + 1$. Hence $\overline{\gamma}_{j+2}$ must notice either γ_i or γ_{i+2}; it then confronts γ_i and γ_{i+2} respectively. ∎

Corollary 7.78 *Let $\textsc{Color}(\gamma_i) = \textsc{Color}(\overline{\gamma}_j) \neq 0$, and let $\textsc{Color}(\gamma_i)$, $\textsc{Color}(\gamma_{i+2})$, and $\textsc{Color}(\overline{\gamma}_{j+2})$ all be different. If γ_i notices $\overline{\gamma}_j$, then there exists s following $fstate(\gamma_i)$ in α such that $s \in \mathcal{C}^1$ and ($s.now < fstate(\gamma_i).now + 42\delta + 20$).*

Proof. From Lemma 7.77, either γ_{i+2} confronts ($\overline{\gamma}_j$ or $\overline{\gamma}_{j+2}$), or $\overline{\gamma}_{j+2}$ confronts (γ_i or γ_{i+2}). Let (s_1, a, s_2) be a "confrontation step" from those mentioned above. From Lemmas 7.70 and 7.71, the durations of γ_i and $\overline{\gamma}_j$ are at most $13\delta + 6$, and those of γ_{i+1} and $\overline{\gamma}_{j+1}$ are at most $3\delta + 2$. Hence $fstate(\gamma_{i+2}).now \leq fstate(\gamma_i).now + (13\delta + 6) + (3\delta + 2)$, and $fstate(\overline{\gamma}_{j+2}).now \leq fstate(\overline{\gamma}_j).now + (13\delta + 6) + (3\delta + 2)$. Since γ_i notices $\overline{\gamma}_j$, Lemma 7.70 implies that $fstate(\overline{\gamma}_j).now \leq fstate(\gamma_i).now + (13\delta + 6)$. Therefore $fstate(\overline{\gamma}_{j+2}).now \leq fstate(\gamma_i).now + (29\delta + 14)$. Lemma 7.72 then implies that there exists a state s following $fstate(\gamma_i)$ in α such that $s \in \mathcal{C}^1$ and $s.now < fstate(\gamma_i).now + (29\delta + 14) + 13\delta + 6$, which yields the result. ∎

7.3.2 The Tree Detection Proposition

Theorem 7.79 $\mathcal{C}_{\overline{W}C}^{=} \xrightarrow[2/9]{58\delta+28} \mathcal{C}^1$.

Proof. Let $s \in \mathcal{C}_{\overline{W}C}^{=}$. If $s \notin \mathcal{C}^1$, then $|s.\Phi| \geq 2$, so there exists more than one tree in s. Let T and $\overline{T}$ be two neighboring trees. Let α be any execution fragment of RSST starting with s, and let α' be the maximal prefix of α that is in $\mathcal{C}_{\overline{W}C}^{=}$. Let α' be partitioned into coloring epochs γ_i for T and $\overline{\gamma}_i$ for $\overline{T}$ such that $\alpha' = \gamma_1 a \gamma_2 a \gamma_3 \ldots = \overline{\gamma}_1 a \overline{\gamma}_2 a \overline{\gamma}_3 \ldots$. By Lemma 7.73, unless a state in $\mathcal{C}^1$ is reached in α before $lstate(\gamma_3)$, there exists $i \leq 3$ such that γ_i notices $\overline{\gamma}_j$ for some j. By Lemmas 7.70 and 7.71, $fstate(\gamma_i).now \leq s.now + (16\delta + 8)$.

If $\mathrm{COLOR}(\gamma_i) \neq \mathrm{COLOR}(\overline{\gamma}_j)$, by Lemma 7.72 there exists state s' in α such that $s' \in \mathcal{C}^1$ and $s'.now < fstate(\gamma_i).now + (13\delta + 6) \leq s.now + (29\delta + 14)$. If $\mathrm{COLOR}(\gamma_i) = \mathrm{COLOR}(\overline{\gamma}_j)$, let (s_1, a, s_2) be a step in which γ_i notices $\overline{\gamma}_j$. Consider the execution automaton $\tilde{H} = H(\mathsf{RSST}, \mathcal{A}, s_1)$.

Let the event e' be defined as the event in which $\mathrm{COLOR}(\gamma_i)$, $\mathrm{COLOR}(\gamma_{i+2})$, and $\mathrm{COLOR}(\overline{\gamma}_{j+2})$ are all different. Then,

$$
\begin{aligned}
P_{\tilde{H}}(e') &= P(\ \mathrm{COLOR}(\gamma_i) \neq \mathrm{COLOR}(\gamma_{i+2})\ ) \times P(\ \mathrm{COLOR}(\overline{\gamma}_{j+2}) \notin \{\mathrm{COLOR}(\gamma_i), \mathrm{COLOR}(\gamma_{i+2})\}\ ) \\
&= 2/3 \times 1/3 \text{ (since the colors are chosen from \{1,2,3\})} \\
&= 2/9
\end{aligned}
$$

For any execution $\alpha \in e'$, Corollary 7.78 implies that there exists s' following $fstate(\gamma_i)$ in α such that $s' \in \mathcal{C}^1$ and $s.now < fstate(\gamma_i).now + 42\delta + 20$. Since $fstate(\gamma_i) \leq s.now + 16\delta + 8$, the Lemma follows. ∎

We are now in a position to state the Tree Detection Proposition:

Proposition 7.80 $\mathcal{C}^{=} \xrightarrow[2/9]{77\delta+36} \mathcal{C}^1$.

Proof. From Lemma 7.66, $\mathcal{C}^{=} \xRightarrow{19\delta+8} \mathcal{C}_{\overline{W}C}^{=}$, and from Lemma 7.79, $\mathcal{C}_{\overline{W}C}^{=} \xrightarrow[2/9]{58\delta+28} \mathcal{C}^1$. Hence the Proposition follows. ∎

105

Chapter 8

The Deterministic Version

In this chapter we describe the main ideas behind the deterministic version of the algorithm, for ID-based networks.

For our deterministic algorithm, we assume that each node has access to a "hardwired" unique ID. We refer to the unique ID as the node's UID to prevent confusion with the nodes "other" ID, which is a tuple of entries as in the randomized case. The "hardwiring" of the UID implies that the UID cannot be corrupted by the adversary; a nodes' UID always remains fixed and unique.

The deterministic protocol is very similar to the randomized version. Each node has an ID consisting of a tuple of entries; each entry is now an integer instead of a pair as for the randomized version. The tree overrunning process (and action MAXIMIZE-PRIORITY) is also identical: nodes attempt to form rooted trees, and trees compete with one another for being the eventual spanning tree.

The main simplification, compared to the randomized version, arises in the method for recoloring trees. We no longer need random coin flips to break symmetry: the unique UIDs are exploited for fully reliable symmetry breaking. Each node, as before, has a *color*. However, the main difference is that trees do *not* need to be repeatedly recolored. The root of a tree always attempts to propagate *its UID as the color of its tree*, so nodes repeatedly copy their

parent's color. If a leaf notices a neighbor with the same ID but a different color, it concludes that its neighbor belongs to a different tree, and informs its root through the *other-trees* variable which is echoed to its root by its ancestors in the tree. When a root detects the presence of a competing tree, it *appends its own UID to its ID*; this change in its ID is automatically propagated to its leaves. Note that we do not need the variables *direction* and *recorded-color* in the deterministic case.

The correctness and complexity proofs are analogous to those for the randomized version, with the exception that all *probabilities* in Chapter 6 are now *certainties*.

Chapter 9

Conclusions and Discussion

In this thesis we have presented self-stabilizing algorithms for constructing spanning trees in asynchronous networks in $O(diameter)$ time; our algorithms are time-optimal. We have presented both a randomized version for anonymous networks and a deterministic version for ID-based networks; both versions use the same general paradigm. We have presented a formal analysis of the randomized protocol using the Probabilistic Automata formalism of Segala and Lynch; in doing so, we have demonstrated the capability of the model to effectively analyze the interactions between the probabilistic choices made by the random algorithmic steps and the nondeterministic choices made by the scheduler.

Besides the stabilization time, another key measure of efficiency (which we have hitherto not dwelt upon) is the *space required at each node*, i.e. the size of the local memory needed at each node to execute the algorithm. The optimal space requirement for an ID-based protocol must necessarily be $\Omega(\log n)$ (since there must exist IDs of size $\Omega(\log n)$).

Our deterministic protocol requires ID extensions of size $O(\log n)$, and our randomized protocol requires extensions of expected size $O(\log \log n)$. Since in a "well-colored" state (cf. Section 7) a root extends only if there exists another root with the same ID, it is likely that each root requires a total of $O(1)$ extensions in both versions of the protocol. If so, both protocols would require space only $O(\log n)$ bits larger than the space occupied at the "start"

of the algorithm. (For the purposes of self-stabilization, the adversary is allowed to set the "initial" state, which might occupy an arbitrary amount of space (since in our protocols IDs can get arbitrarily large). However, the protocols then would "consume" at most expected $O(\log n)$ bits of memory more than the size of the longest "initial" ID.)

A current weakness of our scheme is that it is not guaranteed to function in bounded space; if the adversary sets "too much" of the initial bounded memory, the protocol could run out of space. An important open problem is to construct a time-optimal self-stabilizing spanning tree protocol that runs in bounded space, without any prior knowledge about the network parameters.

Appendix A

Properties of the Afek-Matias Probability Distribution

We now prove Theorems 4.2 and 4.3 stated in Section 4.1. Recall the definitions of Section 4.1. We first prove Theorem 4.2:

Theorem A.1 *For any k, i, $P_{\Gamma^k}(\text{UNIQH} \mid (\text{Highest} > i)) \geq 1/2$.*

For the rest of this chapter, to ease the notation, let U denote the event UNIQH, and let H denote the random variable *Highest*.

Recall that a flip x actually represents a *pair* (s, t), where $P(s = y) = 1/2^y$, and $P(t = y) = 1/\kappa$, where for our purposes $\kappa = 20 \ln 4r$.

We will use the following result throughout this section:

Claim A.2 $P_{\Gamma}(x) = P_{\Gamma}((s, t)) = 1/(2^s \cdot \kappa)$. ∎

Claim A.3 $(a < b) \Longrightarrow (P_{\Gamma}(a) \geq P_{\Gamma}(b))$.

Proof. Let $a = (s_a, t_a)$ and $b = (s_b, t_b)$, and let $a < b$. Then if $s_a < s_b$, $P_{\Gamma}(a) > P_{\Gamma}(b)$. If $s_a = s_b$ and $t_a < t_b$, $P_{\Gamma}(a) = P_{\Gamma}(b)$. ∎

Lemma A.4 *If $a < b$, then*

$$P_\Gamma((X < a) \mid (X \leq a)) \quad < \quad P_\Gamma((X < b) \mid (X \leq b)).$$

Proof. We have,

$$
\begin{aligned}
P_\Gamma((X < a) \mid (X \leq a)) \quad &= \quad \frac{P_\Gamma(X < a)}{P_\Gamma(X \leq a)} \\
&= \quad \frac{P_\Gamma(X \leq a) - P_\Gamma(X = a)}{P_\Gamma(X \leq a)} \\
&= \quad 1 - \frac{P_\Gamma(X = a)}{P_\Gamma(X \leq a)}
\end{aligned}
$$

Similarly,

$$P_\Gamma((X < b) \mid (X \leq b)) = 1 - \frac{P_\Gamma(X = b)}{P_\Gamma(X \leq b)}$$

But clearly $P_\Gamma(X \leq a) \; < P_\Gamma(X \leq b)$, and from Claim A.3, $P_\Gamma(X = a) \geq P_\Gamma(X = b)$. Hence the Lemma follows. ∎

Henceforth, unless otherwise mentioned, all probabilities are assumed to be in the space $,_{AM}^{k}$.

Lemma A.5 *If $a < b$, $P_{\Gamma^k}(U \mid (H = a)) \leq P_{\Gamma^k}(U \mid (H = b))$.*

Proof. In the event $(U \cap (H = a))$ in $,_{AM}^{k}$, the highest of the k flips is unique and is equal to a; all the other $k - 1$ flips are less than a. Hence $P_{\Gamma^k}(U \mid (H = a)) = k \times [P_\Gamma((X < a) \mid (X \leq a))]^{k-1}$, and similarly $P_{\Gamma^k}(U \mid (H = b)) = k \times [P_\Gamma((X < b) \mid (X \leq b))]^{k-1}$. The Lemma follows from Lemma A.4. ∎

Lemma A.6 *For any i, $P(U \mid (H > i)) \geq P(U \mid (H \leq i))$.*

112

Proof. We have,

$$
\begin{aligned}
P(\mathrm{U} \mid (H \leq i)) &= \frac{P(\mathrm{U} \cap (H \leq i))}{P(H \leq i)} \\
&= \frac{P(\mathrm{U} \cap ((H = 1) \cup (H = 2) \cup \ldots \cup (H \leq i)))}{P(H = 1) + P(H = 2) + \ldots + P(H = i)} \\
&= \frac{\sum_{m=1}^{i} P(\mathrm{U} \cap (H = m))}{\sum_{m=1}^{i} P(H = m)} \\
&= \frac{\sum_{m=1}^{i} P(H = m) P(\mathrm{U} \mid (H = m))}{\sum_{m=1}^{i} P(H = m)}
\end{aligned}
\tag{A.1}
$$

Similarly,

$$
P(\mathrm{U} \mid (H > i)) = \frac{\sum_{m=i+1}^{\infty} P(H = m) P(\mathrm{U} \mid (H = m))}{\sum_{m=i+1}^{\infty} P(H = m)}
\tag{A.2}
$$

Now by Lemma A.5, $\max_{m \leq i} P(\mathrm{U} \mid (H = m)) \leq \inf_{m > i} P(\mathrm{U} \mid (H = m))$. Thus, we can choose a z such that

$$
\max_{m \leq i} P(\mathrm{U} \mid (H = m)) \leq z \leq \inf_{m > i} P(\mathrm{U} \mid (H = m))
$$

Then from (A.1), $P(\mathrm{U} \mid (H \leq i)) \leq z$, and from (A.2), $P(\mathrm{U} \mid (H > i)) \geq z$. Hence the Lemma follows. $\blacksquare$

Theorem A.7 $P_{\Gamma^k}(\mathrm{UNIQH} \mid (Highest > i)) \geq 1/2$.

Proof. We have,

$$
\begin{aligned}
P(\mathrm{U}) &= P(\mathrm{U} \cap (H \leq i)) + P(\mathrm{U} \cap (H > i)) \\
&= P(H \leq i) P(\mathrm{U} \mid (H \leq i)) + P(H > i) P(\mathrm{U} \mid (H > i)) \\
&= [f P(H \leq i) + P(H > i)] P(\mathrm{U} \mid (H > i))
\end{aligned}
$$

where $f \leq 1$, because of Lemma A.6. Since $P(H \leq i) + P(H > i) = 1$, we have

$$
P(\mathrm{U}) \leq P(\mathrm{U} \mid (H > i))
$$

113

Since $P(\mathrm{U}) \geq 1/2$ by Theorem 4.1, it follows that $P(\mathrm{U} \mid (H > i)) \geq 1/2$. ∎

We now proceed with the proof of Theorem 4.3, which states that for any k, i, $P_{\Gamma^k}(Highest \neq i)$ $\geq (1 - e^{-1/4}) = 0.22$.

We first prove an ancillary lemma:

Lemma A.8 *For any ϵ such that $0 \leq \epsilon \leq 1/2$, and any $n \geq 0$,*

$$f(\epsilon, n) \triangleq (1 - \epsilon)^n - (1 - 2\epsilon)^n < 0.78$$

Proof. If $(1 - 2\epsilon)^n \geq 1/2$, then $f(\epsilon, n) \leq 1/2$, so the Lemma holds. We now consider the case in which $(1 - 2\epsilon)^n < 1/2$. Since $(1 - 2n\epsilon) \leq (1 - 2\epsilon)^n$, it follows that $(1 - 2n\epsilon) < 1/2$, which implies that $\epsilon > 1/4n$. Thus

$$f(\epsilon, n) \leq (1 - \epsilon)^n < (1 - \frac{1}{4n})^n < e^{-1/4} < 0.78,$$

thus proving the Lemma. ∎

Given a random flip x, let $x.s$ and $x.t$ denote its two fields. Recall that $P_\Gamma(X.s = j) = 1/2^j$.

Claim A.9

$$P_\Gamma(X.s > j) = \frac{1}{2^j}$$

Proof.

$$
\begin{aligned}
P_\Gamma(X.s > j) &= \sum_{m=j+1}^{\infty} P_\Gamma(X.s = m) \\
&= \sum_{m=j+1}^{\infty} \frac{1}{2^m} \\
&= \frac{1}{2^j}
\end{aligned}
$$

114

Corollary A.10

$$P_\Gamma(X.s \leq j) = 1 - \frac{1}{2^j}$$

Corollary A.11

$$P_\Gamma(X.s < j) = 1 - \frac{1}{2^{j-1}}$$

Claim A.12

$$P_{\Gamma^k}(Highest.s < j) = (1 - \frac{1}{2^{j-1}})^k$$

Claim A.13

$$P_{\Gamma^k}(Highest.s > j) = 1 - (1 - \frac{1}{2^j})^k$$

We now prove the main theorem:

Theorem A.14 *For any* k, i, $P_{\Gamma^k}(Highest \neq i) \geq (1 - e^{-1/4}) > 0.22$.

Proof. Let $i.s = j$. Then,

$$
\begin{aligned}
P_{\Gamma^k}(H \neq i) &= P_{\Gamma^k}(H < i) + P_{\Gamma^k}(H > i) \\
&\geq P_{\Gamma^k}(H.s < j) + P_{\Gamma^k}(H.s > j) \\
&= (1 - \frac{1}{2^{j-1}})^k + 1 - (1 - \frac{1}{2^j})^k \\
&= 1 - [(1 - \frac{1}{2^j})^k - (1 - \frac{1}{2^{j-1}})^k]
\end{aligned}
$$

Setting $1/2^j = \epsilon$, the last expression reduces to

$$P_{\Gamma^k}(H \neq i) \geq 1 - [(1 - \epsilon)^k - (1 - 2\epsilon)^k]$$

Since by Lemma A.8 $(1 - \epsilon)^k - (1 - 2\epsilon)^k < 0.78$, the Theorem follows. ∎

References

[AB89]   Yehuda Afek and Geoffrey Brown. Self-stabilization of the alternating bit protocol. In *Proc. 8th Symposium on Reliable Distributed Systems*, October 1989.

[AG90]   Anish Arora and Mohamed G. Gouda. Distributed reset. In *Proc. 10th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 316–331. Springer-Verlag (LNCS 472), 1990.

[AK93]   Sudhanshu Aggarwal and Shay Kutten. Time Optimal Self-Stabilizing Spanning Tree Algorithms. In *Proc. 13th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 400–410. Springer-Verlag (LNCS 761), 1993.

[AKMPV93] Baruch Awerbuch, Shay Kutten, Yishay Mansour, Boaz Patt-Shamir, and George Varghese. Time Optimal Self-Stabilizing Synchronization. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, May 1993.

[AKY90]   Yehuda Afek, Shay Kutten, and Moti Yung. Memory-efficient self-stabilization on general networks. In *Proc. 4th Workshop on Distributed Algorithms*, Italy, September 1990.

[AM89]   Yehuda Afek and Yossi Matias. Simple and Efficient Election Algorithms for Anonymous Networks. In *3rd International Workshop on Distributed Algorithms*, Nice, France, September 1989.

[Ang80]   Dana Angluin. Local and global properties in networks of processes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, May 1980.

[AP90]   Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. In *31st Annual Symposium on Foundations of Computer Science*, 1990.

[APPS92]   Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Mike Saks. Adapting to asynchronous dynamic networks. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 557–570, May 1992.

[APV91]   Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction. In *32nd Annual Symposium on Foundations of Computer Science*, pages 268–277, October 1991.

[APV92]   Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilizing network protocols. Unpublished manuscript, 1992.

[AS88]   Baruch Awerbuch and Michael Sipser. Dynamic networks are as fast as static networks. In *29th Annual Symposium on Foundations of Computer Science*, pages 206–220, October 1988.

[AV91]   Baruch Awerbuch and George Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *32nd Annual Symposium on Foundations of Computer Science*, pages 258–267, October 1991.

[Awe85]   Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, October 1985.

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

[Dij74]   Edsger W. Dijkstra. Self stabilization in spite of distributed control. *Comm. ACM*, 17:643–644, 1974.

[DIM91]   Shlomi Dolev, Amos Israeli, and Shlomo Moran. Uniform self-stabilizing leader election. In *Proc. 5th Workshop on Distributed Algorithms*, pages 167–180, 1991.

[LSS94]   Nancy Lynch, Isaac Saias, and Roberto Segala. Proving Time Bounds for Randomized Distributed Algorithms. To appear in *Proc. 13th Conf. on Principles of Distributed Computing*, August 1994.

[SL94]   Roberto Segala and Nancy Lynch. A model for randomized concurrent systems. Manuscript, 1994

[Var92]   George Varghese. *Self-Stabilization by Local Checking and Correction*. PhD thesis, MIT Lab. for Computer Science, 1992.