Hierarchical Leader Election Algorithm With Remoteness Constraint

Mohamed Tbarka
ENSIAS
University of Oxford

A thesis submitted for the degree of
Software Engineering

2019

This thesis is dedicated to
my grand father
for inspiring me.

Acknowledgements

I would like to thank the all library media specialists for their participation in the survey who supported my work in this way and helped me get results of better quality. I am also grateful to the members of my committee for their patience and support in overcoming numerous obstacles I have been facing through my research

I would like to thank my fellow doctoral students for their feedback, cooperation and of course friendship. In addition I would like to express my gratitude to the staff of INFRES - Telecom ParisTech for the last minute favors.

Nevertheless, I am also grateful to the Mr Petr Kuznetsov for sharing his dissertation woes, and a glimmer of hope for post-dissertation normalcy and Mrs Sanaa ElFkihi for his efficient supervising.

I would like to thank my friends for accepting nothing less than excellence from me. Last but not the least, I would like to thank my family: my parents and to my brothers and sister for supporting me spiritually throughout writing this thesis and my my life in general.

Abstract

A hierarchical algorithm for electing a leaders' hierarchy in an asynchronous network with dynamically changing communication topology is presented including a remoteness's constraint towards each leader. The algorithm ensures that, no matter what pattern of topology changes occur, if topology changes cease, then eventually every connected component contains a unique leaders' hierarchy. The algorithm combines ideas from the Temporally Ordered Routing Algorithm (TORA) for mobile ad hoc networks with a wave algorithm, all within the framework of a height-based mechanism for reversing the logical direction of communication links. Moreover, an improvement from the algorithm in is the introduction of logical clocks as the nodes' measure of time, instead of requiring them to have access to a common global time. This new feature makes the algorithm much more flexible and applicable to real situations, while still providing a correctness proof. It is also proved that in certain well behaved situations, a new leader is not elected unnecessarily.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Leader election is an important primitive for distributed computing, useful as a sub-routine for any application that requires the selection of a unique processor among multiple candidate processors. Applications that need a leader range from the primary-backup approach for replication-based fault-tolerance to group communication systems [26], and from video conferencing to multi-player games [11].

In a dynamic network, communication channels go up and down frequently. Causes for such communication volatility range from the changing position of nodes in mobile networks to failure and repair of point-to-point links in wired networks. Recent research has focused on porting some of the applications mentioned above to dynamic networks, including wireless and sensor networks. For instance, Wang and Wu propose a replication-based scheme for data delivery in mobile and fault-prone sensor networks [29]. Thus there is a need for leader election algorithms that work in dynamic networks.

We consider the problem of ensuring that, if changes to the communication topology cease, then eventually each connected component of the network has a unique leader (introduced as the "local leader election problem" in [7]). Our algorithm is an extension of the leader election algorithm in [18], which in turn is an extension of the MANET routing algorithm TORA in [22]. TORA itself is based on ideas from [9].

Gafni and Bertsekas [9] present two routing algorithms based on the notion of link reversal. The goal of each algorithm is to create directed paths in the communication topology graph from each node to a distinguished destination node. In these algorithms, each node maintains a height variable, drawn from a totally-ordered set; the (bidirectional) communication link between two nodes is considered to be directed from the endpoint with larger height to that with smaller height. Whenever a node becomes a sink, i.e., has no outgoing links, due to a link going down or due to notification of a neighbor's changed height, the node increases its height so that at least one of its incoming links becomes outgoing. In one of the algorithms of [9], the height is a pair consisting of a counter and the node's unique id, while in the other algorithm the height is a triple consisting of two counters and the node id. In both algorithms, heights are compared lexicographically with the least significant component being the node id. In the first algorithm, a sink increases its counter to be larger than the counter of all its neighbors, while in the second algorithm, a more complicated rule is employed for changing the counters.

The algorithms in [9] cause an infinite number of messages to be sent if a portion of the communication graph is disconnected from the destination. This drawback is overcome in TORA [22], through the addition of a clever mechanism by which nodes can identify that they have been partitioned from the destination. In this case, the nodes go into a quiescent state.

In TORA, each node maintains a 5-tuple of integers for its height, consisting of a 3-tuple called the reference level, a delta component, and the node's unique id. The height tuple of each node is lexicographically compared to the tuple of each neighbor to impose a logical direction on links (higher tuple toward lower.)

The purpose of the reference level is to indicate when nodes have lost their directed path to the destination. Initially, the reference level is all zeroes. When a node loses its last outgoing link due to a link going down the node starts a new reference level by changing the first component of the triple to the current time, the second to its own id, and the third to 0, indicating that a search for the destination is started. Reference levels are propagated throughout a connected component, as nodes lose outgoing links due to height changes, in a search for an alternate directed path to the destination. Propagation of reference levels is done using a mechanism by which a node increases its reference level when it becomes a sink; the delta value of the height is manipulated to ensure that links are oriented appropriately. If the search in one part of the graph is determined to have reached a dead end, then the third component of the reference level triple is set to 1. When this happens, the reference level is said to have been reflected, since it is subsequently propagated back toward the originator. If the originator receives reflected reference levels back from all its neighbors, then it has identified a partitioning from the destination.The purpose of the reference level is to indicate when nodes have lost their directed path to the destination. Initially, the reference level is all zeroes. When a node loses its last outgoing link due to a link going down the node starts a new reference level by changing the first component of the triple to the current time, the second to its own id, and the third to 0, indicating that a search for the destination is started. Reference levels are propagated throughout a connected component, as nodes lose outgoing links due to height changes, in a search for an alternate directed path to the destination. Propagation of reference levels is done using a mechanism by which a node increases its reference level when it becomes a sink; the delta value of the height is manipulated to ensure that links are oriented appropriately. If the search in one part of the graph is determined to have reached a dead end, then the third component of the reference level triple is set to 1. When this happens, the reference level is said to have been reflected, since it is subsequently propagated back toward the originator. If the originator receives reflected reference levels back from all its neighbors, then it has identified a partitioning from the destination.

The key observation in [18] is that TORA can be adapted for leader election: when a node detects that it has been partitioned from the old leader (the destination), then, instead of becoming quiescent, it elects itself. The information about the new leader is then propagated through the connected component. A sixth component was added to the height tuple of TORA to record the leader's id. The algorithm presented and analyzed in [18] makes several strong assumptions. First, it is assumed that only one topology change occurs at a time, and no change occurs until the system has finished reacting to the previous change. In fact, a scenario involving multiple topology changes can be constructed in which the algorithm is incorrect. Second, the system is assumed to be synchronous; in addition to nodes having perfect clocks, all messages have a fixed delay. Third, it is assumed that the two endpoints of a link going up or down are notified simultaneously of the change.

We present a modification to the algorithm that works in an asynchronous system with arbitrary topology changes that are not necessarily reported instantaneously to both endpoins of a link. One new feature of this algorithm is to add a seventh component to the height tuple of [18]: a timestamp associated with the leader id that records the time that the leader was elected. Also, a new rule by which nodes can choose new leaders is included. A newly elected leader initiates a "wave" algorithm [27]:

when different leader ids collide at a node, the one with the most recent timestamp is chosen as the winner and the newly adopted height is further propagated. This strategy for breaking ties between competing leaders makes the algorithm compact and elegant, as messages sent between nodes carry only the height information of the sending node, every message is identical in structure, and only one message type is used.

In this paper, we relax the requirement in [18] (and in [15]) that nodes have perfect clocks. Instead we use a more generic notion of time, a causal clock $\mathcal{T}$, to represent any type of clock whose values are non-negative real numbers and that preserves the causal relation between events. Both logical clocks [16] and perfect clocks are possible implementations of $\mathcal{T}$. We also relax the requirement in [18] (and in [15]) that the underlying neighbor-detection layer synchronize its notifications to the two endpoints of a (bidirectional) communication link throughout the execution; in the current paper, these notifications are only required to satisfy an eventual agreement property.

Finally, we provide a relatively brief, yet complete, proof of algorithm correctness. In addition to showing that each connected component eventually has a unique leader, it is shown that in certain well-behaved situations, a new leader is not elected unnecessarily; we identify a set of conditions under which the algorithm is "stable" in this sense. We also compare the difference in the stability guarantees provided by the perfect-clocks version of the algorithm and the causal-clocks version of the algorithm. The proofs handle arbitrary asynchrony in the message delays, while the proof in [18] was for the special case of synchronous communication rounds only and did not address the issue of stability.

Leader election has been extensively studied, both for static and dynamic networks, the latter category including mobile networks. Here we mention some representative papers on leader election in dynamic networks. Hatzis et al. [12] presented algorithms for leader election in mobile networks in which nodes are expected to control their movement in order to facilitate communication. This type of algorithm is not suitable for networks in which nodes can move arbitrarily. Vasudevan et al. [28] and Masum et al. [20] developed leader election algorithms for mobile networks with the goal of electing as leader the node with the highest priority according to some criterion. Both these algorithms are designed for the broadcast model. In contrast, our algorithm can elect any node as the leader, involves fewer types of messages than either of these two algorithms, and uses point-to-point communication rather than broadcasting. Brunekreef et al. [2] devised a leader election algorithm for a 1-hop wireless environment in which nodes can crash and recover. Our algorithm is suited to an arbitrary communication topology.

Several other leader election algorithms have been developed based on MANET routing algorithms. The algorithm in [23] is based on the Zone Routing Protocol [10]. A correctness proof is given, but only for the synchronous case assuming only one topology change. In [5], Derhab and Badache present a leader election algorithm for ad hoc wireless networks that, like ours, is based on the algorithms presented by Malpani et al. [18]. Unlike Derhab and Badache, we prove our algorithm is correct even when communication is asynchronous and multiple topology changes, including network partitions, occur during the leader election process.

Dagdeviren et al. [3] and Rahman et al. [24] have recently proposed leader election algorithms for mobile ad hoc networks; these algorithms have been evaluated solely through simulation, and lack correctness proofs. A different direction is randomized leader election algorithms for wireless networks (e.g., [1]); our algorithm is deterministic.

Fault-tolerant leader election algorithms have been proposed for wired networks. Representative examples are Mans and Santoro's algorithm for loop graphs subject to permanent communication failures [19], Singh's algorithm for complete graphs subject to intermittent communication failures [25], and Pan and Singh's algorithm [21] and Stoller's algorithm [26] that tolerate node crashes.

Recently, Datta et al. [4] presented a self-stabilizing leader election algorithm for the shared memory model with composite atomicity that satisfies stronger stability properties than our causal-clocks algorithm. In particular, their algorithm ensures that, if multiple topology changes occur simultaneously after the algorithm has stabilized, and then no further changes occur, (1) each node that ends up in a connected component with at least one pre-existing leader ultimately chooses a pre-existing leader, and (2) no node changes its leader more than once. The self-stabilizing nature of the algorithm suggests that it can be used in a dynamic network: once the last topology change has occurred, the algorithm starts to stabilize. Existing techniques (see, for instance, Section 4.2 in [6]) can be used to transform a self-stabilizing algorithm for the shared-memory composite-atomicity model into an equivalent algorithm for a (static) message-passing model, perhaps with some timing information. Such a sequence of transformations, though, produces a complicated algorithm and incurs time and space overhead (cf. [6,13]). One issue to be overcome in transforming an algorithm for the static message-passing model to the model in our paper is handling the synchrony that is relied upon in some component transformations to message passing (e.g., [14]).

Chapter 2

Preliminaries

2.1 System Model

We assume a system consisting of a set P of computing nodes and a set χ of directed communication channels from one node to another node. χ consists of one channel for each ordered pair of nodes, i.e., every possible channel is represented. The nodes are assumed to be completely reliable. The channels between nodes go up and down, due to the movement of the nodes. While a channel is up, the communication across it is in first-in-first-out order and is reliable but asynchronous (see below for more details).

We model the whole system as a set of (infinite) state machines that interact through shared events (a specialization of the IOA model [17]). Each node and each channel is modeled as a separate state machine. The events shared by a node and one of its outgoing channels are notifications that the channel is going up or going down and the sending of a message by the node over the channel; the channel up/down notifications are initiated by the channel and responded to by the node, while the message sends are initiated by the node and responded to by the channel. The events shared by a node and one of its incoming channels are notifications that a message is being delivered to the node from the channel; these events are initiated by the channel and responded to by the node.

2.2 Modeling Asynchronous Dynamic Links

We now specify in more detail how communication is assumed to occur over the dynamic links. The state of $Channel(u, v)$, which models the communication channel from node u to node v, consists of a statusuv variable and a queue *mqueueuv* of messages.

The possible values of the $status_{uv}$ variable are *Up* and *Down*. The channel transitions between the two values of its $status_{uv}$ variable through $ChannelUp_{uv}$ and $ChannelDown_{uv}$ events, called the "topology change" events. We assume that the *ChannelUp* and *ChannelDown* events for the channel alternate. The *ChannelUp* and *ChannelDown* events for the channel from u to v occur simultaneously at node u and the channel, but do not occur at node v.

The variable $mqueue_{uv}$ holds messages in transit from u to v. An attempt by node u to send a message to node v results in the message being appended to $mqueue_{uv}$ if the channel's status is *Up*; otherwise there is no effect. When the channel is *Up*, the message at the head of $mqueue_{uv}$ can be delivered to node v; when a message is delivered, it is removed from $mqueue_{uv}$. Thus, messages are delivered in FIFO order.

When a $ChannelDown_{uv}$ event occurs, $mqueue_{uv}$ is emptied. Neither u nor v is alerted to which messages in transit have been lost. Thus, the messages delivered to node v from node u during a (maximal-length) interval when the channel is *Up* form a prefix of the messages sent by node u to node v during that interval.

2.3 Configurations and Executions

The notion of configuration is used to capture an instantaneous snapshot of the state of the entire system. A configuration is a vector of node states, one for each node in $\mathcal{P}$, and a vector of channel states, one for each channel in χ . In an initial configuration: spacing

- each node is in an initial state (according to its algorithm),
- for each channel $Channel(u, v)$, $mqueue_{uv}$ is empty, and
- for all nodes u and v, $status_{uv} = status_{vu}$ (i.e., either both channels between u and v are up, or both are down).

Define an execution as an infinite sequence $C_0, e_1, C_1, e_2, C_2, ...$ of alternating configurations and events, starting with an initial configuration and, if finite, ending with a configuration such that the sequence satisfies the following conditions: spacing

– C_0 is an initial configuration.

– The preconditions for event ei are true in C_{i-1} for all $i \geq 1$.

– C_i is the result of executing event e_i on configuration C_{i-1} , for all $i \geq 1$ (only the node and channel involved in an event change state, and they change according to their state machine transitions).

– If a channel remains Up for infinitely long, then every message sent over the channel during this Up interval is eventually delivered.

– For all nodes u and v, Channel(u, v) experiences infinitely many topology change events if and only if Channel(v, u) experiences infinitely many topology change events; if they both experience finitely many, then after the last one, $status_{uv} = status_{vu}$.

Given a configuration of an execution, define an undirected graph G_{chan} as follows: the vertices are the nodes, and there is an (undirected) edge between vertices u and v if and only if at least one of $Channel_{uv}$ and $Channel_{vu}$ is Up. Thus G_{chan} indicates all pairs of nodes u and v such that either u can send messages to v or v can send messages to u. If the execution has a finite number of topology change events, then G_{chan} never changes after the last such event, and we denote the final version of final final G_{chan} as G_{chan} . By the last bullet point above, an edge in G_{chan} indicates bidirectional communication ability between the two endpoints.

We also assign a positive real-valued global time gt to each event e_i , $i \geq 1$, such that $gt(ei) < gt(e_{i+1})$ and, if the execution is infinite, the global times increase without bound. Each configuration inherits the global time of its preceding event, so $gt(C_i) = gt(e_i)$ for $i \geq 1$; we define $gt(C_0)$ to be 0. We assume that the nodes do not have access to gt.

Instead, each node u has a causal clock $\mathcal{T}_u$, which provides it with a non-negative real number at each event in an execution. $\mathcal{T}_u$ is a function from global time (i.e., positive reals) to causal clock times; given an execution, for convenience we sometimes use the notation $\mathcal{T}_u(e_i)$ or $\mathcal{T}_u(C_i)$ as shorthand for $\mathcal{T}_u(gt(e_i))$ or $\mathcal{T}_u(gt(C_i))$. The key idea of causal clocks is that if one event potentially can cause another event, then the clock value assigned to the first event is less than the clock value assigned to the second event. We use the notion of happens-before to capture the concept of potential causality. Recall that an event e_1 is defined to happen before [16] another event e_2 if one of the following conditions is true:

1. Both events happen at the same node, and e_1 occurs before e_2 in the execution.
2. e_1 is the send event of some message from node u to node v, and e_2 is the receive event of that message by node v.
3. There exists an event e such that e_1 happens before e and e happens before e_2.

The causal clocks at all the nodes, collectively denoted T , must satisfy the following properties: spacing

– For each node u, the values of $\mathcal{T}_u$ are increasing, i.e., if e_i and e_j are events involving u in the execution with $i < j$, then $\mathcal{T}_u(e_i) < \mathcal{T}_u(e_j)$. In particular, if there is an infinite number of events involving u, then $\mathcal{T}_u$ increases without bound.

– $\mathcal{T}$ preserves the happens-before relation [16] on events; i.e., if event e_i happens before event e_j, then $\mathcal{T}(e_i) < \mathcal{T}(e_j)$.

Our description and proof of the algorithm assume that nodes have access to causal clocks. One way to implement causal clocks is to use perfect clocks, which ensure that $\mathcal{T}_u(t) = t$ for each node u and global time t. Since an event that causes another event must occur before it in real time, perfect clocks capture causality. Perfect clocks could be provided by, say a GPS service, and were assumed in the preliminary version of this paper [15]. Another way to implement causal clocks is to use Lamport's logical clocks [16], which were specifically designed to capture causality.

2.4 Problem Definition

Each node u in the system has a local variable lid_u to hold the identifier of the node currently considered by u to be the leader of the connected component containing u.

In every execution that includes a finite number of topology change events, we require that the following eventually holds: Every connected component CC of the final final topology graph G_chan contains a node l, the leader, such that $lid_u = l$ for all nodes $u \in CC$, including l itself.

Chapter 3

Leader Election Algorithm

In this section, we present our leader election algorithm. The pseudocode for the algorithm is presented in Figures 1, 2 and 3. First, we provide an informal description of the algorithm, then, we present the details of the algorithm and the pseudocode, and finally, we provide an example execution. In the rest of this section, variable var of node u will be indicated as varu . For brevity, in the pseudocode for node u, variable varu is denoted by just var.

3.1 Informal Description

Each node in the system has a 7-tuple of integers called a height. The directions of the edges in the graph are determined by comparing the heights of neighboring nodes: an edge is directed from a node with a larger height to a node with a smaller height. Due to topology changes nodes may lose some of their incident links, or get new ones throughout the execution. Whenever a node loses its last outgoing link because of a topology change, it has no path to the current leader, so it reverses all of its incident edges. Reversing all incident edges acts as the start of a search mechanism (called a reference level) for the current leader. Each node that receives the newly started reference level reverses the edges to some of its neighbors and in effect propagates the search throughout the connected component. Once a node becomes a sink and all of its neighbors are already participating in the same search, it means that the search has hit a dead end and the current leader is not present in this part of the connected component. Such dead-end information is then propagated back towards the originator of the search. When a node which started a search receives such dead-end messages from all of its neighbors, it concludes that the current leader is not present in the connected component, and so the originator of the search elects itself as the new leader. Finally, this new leader information propagates throughout the network via an extra "wave" of propagation of messages.

In our algorithm, two of the components of a node's height are timestamps recording the time when a new "search" for the leader is started, and the time when a leader is elected. In the algorithm in [15], these timestamps are obtained from a global clock accessible to all nodes in the system. In this paper, we use the notion of causal clocks (defined in Section 2.3) instead.

One difficulty that arises in solving leader election in dynamic networks is dealing with the partitioning and merging of connected components. For example, when a connected component is partitioned from the current leader due to links going down, the above algorithm ensures that a new leader is elected using the mechanism of waves searching for the leader and convergecasting back to the originator. On the other hand, it is also possible that two connected components merge together resulting in two leaders in the new connected component. When the different heights of the two leaders are being propagated in the new connected component, eventually, some node needs to compare both and decide which one to adopt and continue propagating. Recall that when a new leader is elected, a component of the height of the leader records the time of the election which can be used to determine the more recent of two elections. Therefore, when a node receives a height with a different leader information from its own, it adopts the one corresponding to the more recent election.

Similarly, if two reference levels are being propagated in the same connected component, whenever a node receives a height with a reference level different from its current one, it adopts the reference level with the more recent timestamp and continues propagating it. Therefore, even though conflicting information may be propagating in the same connected component, eventually the algorithm ensures that as long as topology changes stop, each connected component has a unique leader.

3.2 Nodes, Neighbors and Heights

First, we describe the mechanism through which nodes get to know their neighbors. Each node in the algorithm keeps a directed approximation of its neighborhood in Gchan as follows. When u gets a ChannelUp event for the channel from u to v, it puts v in a local set variable called formingu . When u subsequently receives a message from v, it moves v from its formingu set to a local set variable called Nu (N for neighbor). If u gets a message from a node which is neither in its forming set, nor in Nu , it ignores that message. And when u gets a ChannelDown event for the channel from u to v, it removes v from formingu or Nu , as appropriate. For the purposes of the algorithm, u considers as its neighbors only those nodes in Nu . It is possible for two nodes u and v to have inconsistent views concerning whether u and v are neighbors of each other. We will refer to the ordered pair (u, v), where v is in Nu , as a link of node u.

Nodes assign virtual directions to their links using variables called heights. Each node maintains a height for itself, which can change over time, and sends its height over all outgoing channels at various points in the execution. Each node keeps track of the heights it has received in messages. For each link (u, v) of node u, u considers the link as incoming (directed from v to u) if the height that u has recorded for v is larger than u's own height; otherwise u considers the link as outgoing (directed from u to v). Heights are compared using lexicographic ordering; the definition of height ensures that two nodes never have the same height. Note that, even if v is viewed as a neighbor of u and vice versa, u and v might assign opposite directions to their corresponding links, due to asynchrony in message delays.

Next, we examine the structure of a node's height in more detail. The height for each node is a 7-tuple of integers $((\tau, oid, r), \delta, (nlts, lid), id)$, where the first three components are referred to as the reference level (RL) and the fifth and sixth components are referred to as the leader pair (LP). In more detail, the components are defined as follows: spacing

- τ , a non-negative timestamp which is either 0 or the value of the causal clock time when the current search for an alternate path to the leader was initiated.
- oid, is a non-negative value that is either 0 or the id of the node that started the current search (we assume node ids are positive integers).
- r, a bit that is set to 0 when the current search is initiated and set to 1 when the current search hits a dead end.
- δ , an integer that is set to ensure that links are directed appropriately to neighbors with the same first three components. During the execution of the algorithm δ serves multiple purposes. When the algorithm is in the stage of

searching for the leader (having either reflected or unreflected RL), the δ value ensures that as a node u adopts the new reference level from a node v, the direction of the edge between them is from v to u; in other words it coincides with the direction of the search propagation. Therefore, u adopts the RL of v and sets its δ to one less than v's. When a leader is already elected, the δ value helps orient the edges of each node towards the leader. Therefore, when node u receives information about a new leader from node v, it adopts the entire height of v and sets the δ value to one more than v's. $--$ $nlts$, a non-positive timestamp whose absolute value is the causal clock time when the current leader was elected. –lid, the id of the current leader. $--$ id, the node's unique ID.

Each node u keeps track of the heights of its neighbors in an array $height_u$, where the height of a neighbor node v is stored in $height_u[v]$. The components of $height_u[v]$ are referred to as $(\tau^v, oid^v, r^v, \delta^v, nlts^v, lid^v, v)$ in the pseudocode.

3.3 Initial States

The definition of an initial configuration for the entire system from Section 2.3 included the condition that each node be in an initial state according to its algorithm. The collection of initial states for the nodes must be consistent with the collection of initial states for the channels. Let G_{init} chan be the undirected graph corresponding to the initial states of the channels, as defined in Section 2.3. Then in an initial configuration, the state of each node u must satisfy the following: spacing

- formingu is empty, – Nu equals the set of neighbors of u in Ginit chan
- heightu[u] = $(0, 0, 0, \delta_u, 0, l, u)$ where l is the id of a fixed node in $u's$ connected chan (the current leader), and δ_u equals the distance from u to l in component in Ginit Ginit chan ,
- for each v in Nu , heightu[v] = heightv [v] (i.e., u has accurate information about v's height), and
- Tu is initialized properly with respect to the definition of causal clocks.

The constraints on the initial configuration just given imply that initially, each connected component of the communication topology graph has a leader; furthermore, by following the virtual directions on the links, nodes can easily forward information to the leader (as in TORA). One way of viewing our algorithm is that it maintains leaders in the network in the presence of arbitrary topology changes. In order to establish this property, the same algorithm can be executed, with each node initially being in a singleton connected component of the topology graph prior to any ChannelUp or ChannelDown events.

3.4 Goal of the Algorithm

The goal of the algorithm is to ensure that, once topology changes cease, eventually each connected component of G_{chan}^{final} is "leader-oriented", which we now define. Let CC be any connected component of G_{chan}^{final} . First, we define a directed version of CC, denoted $CC^{\longleftrightarrow}$, in which each undirected edge of CC is directed from the endpoint with larger height to the endpoint with smaller height. We say that CC is leader-oriented if the following conditions hold:

1. No messages are in transit in CC.
2. For each (undirected) edge u, v in CC, if (u, v) is a link of u, then u has the correct view of v's height.
3. Each node in CC has the same leader id, say l, where l is also in $CC^{\longrightarrow}$.
4. CC is a directed acyclic graph (DAG) with l as the unique sink.

A consequence of each connected component being leader-oriented is that the leader election problem is solved.

3.5 Description of the Algorithm

The algorithm consists of three different actions, one for each of the possible events that can occur in the system: a channel going up, a channel going down, and the receipt of a message from another node. Next, we describe each of these actions in detail.

First, we formally define the conditions under which a node is considered to be a sink: spacing

- SINK = $((\forall v \in N_u, LP_u^v = LP_u^u)$ and $(\forall v \in N_u, height_u[u] < height_u[v])$ and $(lid_u^u = u))$. Recall that the LP component of node $u's$ view of $v's$ height, as stored in $u's$ height array, is denoted LP_u^v , and similarly for all the other height components. This predicate is true when, according to $u's$ local state, all of $u's$ neighbors have the same leader pair as u, u has no outgoing links, and u is not its own leader. If node u has links to any neighbors with different LPs, u is not considered a sink, regardless of the directions of those links.

$ChannelDown$ event: When a node u receives a notification that one of its incident channels has gone down, it needs to check whether it still has a path to the current leader. If the $ChannelDown$ event has caused u to lose its last neighbor, as indicated by u's N variable, then u elects itself by calling the subroutine $ELECTSELF$. In this subroutine, node u sets its first four components to 0, and the LP component to $(nlts, u)$ where $nlts$ is the negative value of u's current causal clock time. Then, in case u has any incident channels that are in the process of forming, u sends its new height over them. If the $ChannelDown$ event has not robbed u of all its neighbors (as indicated by $u'sN$ variable) but u has lost its last outgoing link, i.e., it passes the SINK test, then u starts a new reference level (a search for the leader) by setting its τ value to the current clock time, oid to u's id, the r bit to 0, and the δ value to 0, as shown in subroutine $STARTNEWREFLEVEL$. The complete pseudo-code for the $ChannelDown$ action is available in Figure 1.

$ChannelUp$ event: When a node u receives a notification of a channel going up to another node, say v, then u sends its current height to v and includes v in its set $forming_u$. The pseudo-code for the $ChannelUp$ action is available in Figure 1.

Algorithm 1 When $ChannelDown_{uv}$ event occurs:

```
1: N := N \ v
2: forming := forming \ v
3: if N = ∅ then
4:     ELECTSELF
5:     send Update(height[u]toallw ∈ forming)
6: else if SINK then
7:     STARTNEWREFLEVEL
8:     send Update(height[u]) to all w ∈ (N ∪ forming))
9: end if
```

Algorithm 2 When $ChannelUp_{uv}$ event occurs:

```
1: forming := forming ∪ v
2: send Update(height[u]) to v
```

Receipt of an update message: When a node u receives a message from another node v, containing v's height, node u performs the following sequence of rules (shown in Figure 2).

First, if v is in neither $forming_u$ nor N_u , then the message is ignored. If $v \in forming_u$ but $v \ni N_u$ then v is moved to N_u. Next, u checks whether v has the same leader pair as u. If v knows about a more recent leader than u, node u adopts that new LP (shown in subroutine $ADOPTLPIFPRIORITY$ in Figure 3). If the LP's of u and v are the same, then u checks whether it is a sink using the definition above. If it is not a sink, it does not perform any further action (because it already has a path to the leader). Otherwise, if u is a sink, it checks the value of the RL component of all of its neighbors' heights (including v's). If some neighbor of u, say w, knows of a RL which is more recent than u's, then u adopts that new RL by setting the RL part of its height to the new RL value and changing the δ component to one less than the δ component of w. Therefore, the change in u's height does not cause w to become a sink (again) and so the search for the leader does not go back to w and it is thus prop-agated in the rest of the connected component. The details are shown in subroutine PROPAGATE L ARGEST R EF L EVEL in Figure 3.

If u and all of its neighbors have the same RL component of their heights, say (τ, oid, r), we consider three possible cases:

1. If $\tau > 0$ (indicating that this is a RL started by some node, and not the default value 0) and r = 0 (the RL has not reached a dead end), then this is an indication of a dead end because u and all of its neighbors have the same unreflected RL. In this case u changes its height by setting the r component of its height to 1 (shown in subroutine REFLECT R EF L EVEL in Figure 3).

2. If $\tau > 0$ (indicating that this is a RL started by some node, and not the default value 0), r = 1 (the RL has already reached a dead end) and oid = u (u started the current RL), then this is an indication that the current leader may not be in the same connected component anymore. In other words, all the branches of the RL started by u reached dead ends. Therefore, u elects itself as the new leader by setting its first 4 components to 0, and the LP component to (nlts, u) where nlts is the negative value of u's current causal clock time (shown in subroutine ELECT S ELF in Figure 3). Note that this case does not guarantee that the old leader is not in the connected component, because some recent

topology change may have reconnected it back to u's component. We already described how the leader information of two different leaders is handled.

3. If neither of the two conditions above are satisfied, then it is the case that either $\tau = 0$ or $\tau > 0$, r = 1 and oid 6= u. In other words, all of u's neighbors have a different reflected RL or contain an RL indicating that various topology changes have interfered with the proper propagation of RL's, and so node u starts a fresh RL by setting τ to the current causal clock time, oid to u's id, the r bit to 0, and the δ value to 0 (shown in subroutine STARTNEWREFLEVEL in Figure 3).

Finally, whenever a node changes its height, it sends a message with its new height to all of its neighbors. Additionally, whenever a node u receives a message from a node v indicating that v has different leader information from u, then either if u adopts v's LP or not, u sends an update message to v with its new (possibly same as old) height. This step is required due to the weak level of coordination in neighbor discovery.

3.6 Sample execution

Next, we provide an example which illustrates a particular algorithm execution. Figure 4, parts (a)-(h), show the main stages of the execution. In the picture for each stage, a message in transit over a channel is indicated by a light grey arrow. The recipient of the message has not yet taken a step and so, in its view, the link is not yet reversed.

a A quiescent network is a leader-oriented DAG in which node H is the current leader. The height of each node is displayed in parenthesis. Link direction in this figure is shown using solid-headed arrows and messages in transit are indicated by light grey arrows.

b The link between nodes G and H goes down triggering action ChannelDown at node G (and node H). When non-leader node G loses its last outgoing link due to the loss of the link to node H, G executes subroutine START N EW R EF L EVEL (because it is a sink and it has other neighbors besides H), and sets the RL and δ parts of its height to (1,G,0) and $\delta = 0$. Then node G sends messages with its new height to all its neighbors. By raising its height in this way, G has started a search for leader H.

c Nodes D, E, and F receive the messages sent from node G, messages that cause each of these nodes to become sinks because G's new RL causes its incident edges to be directed away from G. Next, nodes D, E, and F compare their neigh- bors' RL's and propagate G's RL (since nodes B and C have lower heights than node G) by executing PROPAGATE L ARGEST R EF L EVEL. Thus, they take on RL (1,G,0) and set their δ values to -1, ensuring that their heights are lower than G's but higher than the other neighbors'. Then D, E and F send messages to their neighbors.

d Node B has received messages from both E and D with the new RL (1,G,0), and C has received a message from F with RL (1,G,0); as a result, B and C execute subroutine PROPAGATE L ARGEST R EF L EVEL, which causes them to take on RL (1,G,0) with δ set to -2 (they propagate the RL because it is more recent than all of their neighbors' RL's), and send messages to their neighbors.

e Node A has received message from both nodes B and C. In this situation, node A is connected only to nodes that are participating in the search started by node G for leader H. In other words, all neighbors of node A have the same RL with $\delta > 0$ and $r = 0$, which indicates that A has detected a dead end for this search. In this case, node A executes subroutine REFLECT R EF L EVEL, i.e., it "reflects" the search by setting the reflection bit in the $(1, G, *)$ reference level to 1, resetting its δ to 0, and sending its new height to its neighbors.

f Nodes B and C take on the reflected reference level (1,G,1) by executing subroutine PROPAGATE L ARGEST R EF L EVEL (because this is the largest RL among their neighbors) and set their δ to -1, causing their heights to be lower than A's and higher than their other neighbors'. They also send their new heights to their neighbors.

g Nodes D, E, and F act similarly as B and C did in part (f), but set their δ values to -2.

h When node G receives the reflected reference level from all its neighbors, it knows that its search for H is in vain. G executes subroutine ELECT S ELF and elects itself by setting the LP part of its height to $(-7, G)$ assuming the causal clock value at node G at the time of the election is 7. The new LP $(-7, G)$ then propagates through the component, assuming no further link changes occur. Whenever a node receives the new LP information, it adopts it because it is more recent than the one associated with the old LP of H. Eventually, each node has RL (0,0,0) and LP $(-7, G)$, with D, E and F having $\delta = 1$, B and C having $\delta = 2$, and A having $\delta = -3$.

We now explain two other aspects of the algorithm that were not exercised in the example execution just given. First, note that it is possible for multiple searches—each initiated by a call to START N EW R EF L EVEL—for the same leader to be going on simultaneously. Suppose messages on behalf of different searches meet at a node i. We assume that messages are taken out of the input message queue one at a time. Major action is only taken by node i when it loses its last outgoing link; when the ear- lier messages are processed, all that happens is that the appropriate height variables are updated. If and when a message is processed that causes node i to lose its last out- going link, then i takes appropriate action, either to propagate the largest reference level among its neighbors or to reflect the common reference level.

Another potentially troublesome situation is when, for two nodes u and v, the channel from u to v is up for a long period of time while the channel from v to u is down. When the channel from u to v comes up at u, v is placed in u's forming set, but is not able to move into u's neighbor set until u receives an Update message from v, which does not occur as long as the channel from v to u remains down. Thus during this interval, u sends update messages to v but since v is not considered a neighbor of u, v is ignored in deciding whether u is a sink. In the other direction, when the channel from u to v comes up at u, u sends its height to v, but the message is ignored by v since the link from v to u is down and thus u is not in v's forming set or neighbor set. More discussion of this asymmetry appears in Section 4.1; for now, the main point is that the algorithm simply continues with u and v not considering each other as neighbors.

Algorithm 3 When node u receives $Update(h)$ from node $v \in forming \cup N$:

$height[v] := h$ ▷ if v is in neither forming nor N, message is ignored
$forming := forming \setminus v$
$N := N \cup v$
$myOldHeight := height[u]$
if $(nlts_u, lid_u) = (nlts_v, lid_v)$ **then** ▷ leader pair are the same
  if $SINK$ **then**
    if $(\exists(\tau, oid, r)|(\tau_w, oid_w, r_w) = (\tau, oid, r)\forall w \in N$ **then**
      if $(\tau > 0)and(r = 0)$ **then**
        $REFLECTREFLEVEL$
      else if $(\tau > 0)and(r = 1)and(oid = u)$ **then**
        $ELECTSELF$
      else ▷ $(\tau = 0)or(\tau > 0\ and\ r = 1\ and\ oid = u)$
        $STARTNEWREFLEVEL$
      end if
    else ▷ neighbors have different ref levels
      $PROPAGATELARGESTREFLEVEL$
    end if
  end if ▷ else not sink, do nothing
else ▷ leader pairs are different
  $ADOPTLPIFPRIORITY(v)$
end if
if $myOldHeight \neq height[u]$ **then**
  send Update($height[u]$) to all $w \in (N \cup forming)$
end if

Algorithm 4 $ELECTSELF$

$height[u] := (0, 0, 0, 0, -\mathcal{T}_u, u, u)$

Algorithm 5 REFLECTREFLEVEL

$height := (\tau, oid, 1, 0, nlts_u, lid_u, u)$

Algorithm 6 $ELECTSELF$

$height[u] := (0, 0, 0, 0, -\mathcal{T}_u, u, u)$

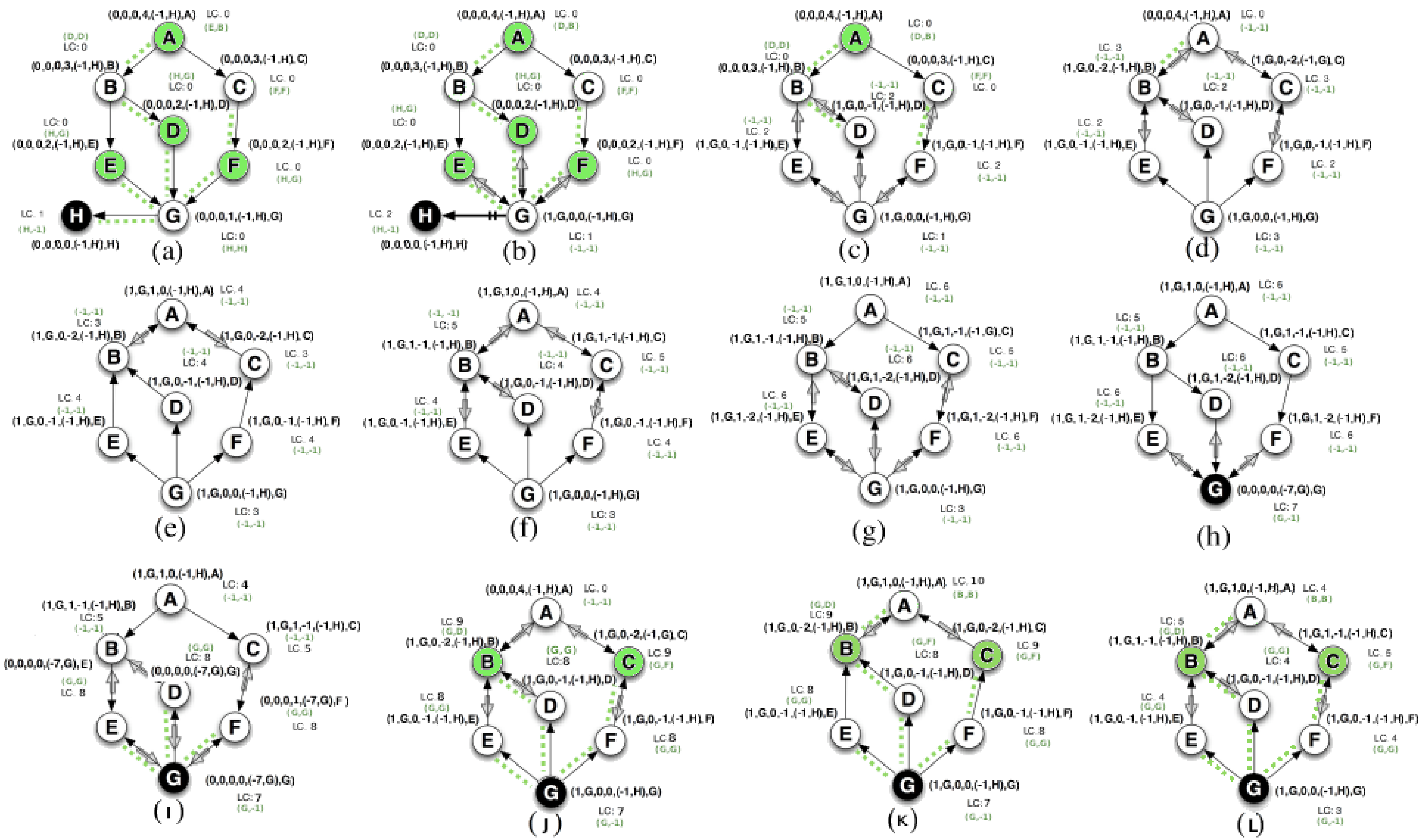

Figure 3.1:

Chapter 4

Correctness

In this section, we show that, once topology changes cease, the algorithm eventually terminates with each connected component being leader-oriented. As a result, the lidu variables satisfy the conditions of the leader election problem.

We first show, in Section 4.1, an important relationship between the final communication topology and the forming and N variables of the nodes. The rest of the proof uses a number of invariants, denoted as "Properties", which are shown to hold in every configuration of every execution; each one is proved (separately) by induction on the configurations occurring in an execution. In Section 4.2, we introduce some definitions and basic facts regarding the information about nodes' heights that appears in the system, either in nodes' height arrays or in messages in transit. In Section 4.3, we bound, in Lemma 3, the number of elections that can occur after the last topology change; this result relies on the fact, shown in Lemma 2, that once a node u adopts a leader that was elected after the last topology change, u never becomes a sink again. Then in Section 4.4, we bound, in Lemma 4, the number of new reference levels that are started after the last topology change; the proof of this result relies on several additional properties. Section 4.5 is devoted to showing, in Lemmas 5, 6, and 7, that eventually there are no messages in transit and every node has an accurate view of its neighbors' heights. All the pieces are put together in Theorem 1 of Section 4.6 to show that eventually we have a leader-oriented connected component; a couple of additional properties are needed for this result.

Throughout the proof, consider an arbitrary execution of the algorithm in which the last topology change event occurs at some global time tLTC , and consider any connected component of the final topology.

4.1 Channels and Neighbors

Because of the lack of coordination between the topology change events for the two channels going between nodes u and v in the two directions, u and v do not neces-sarily have consistent views of their local neighborhoods in Gchan , even after the last topology change. For instance, it is possible that v is in Nu but u is not in Nv forever after the last topology change. Suppose the channel from u to v remains Up from some time t onwards, so that v remains in Nu from time t onwards. However, suppose that the channel from v to u fluctuates several times after time t, eventually stabilizing to being Up (cf. Fig. 5). Every time the channel to u goes down, u is removed from v's forming and N sets. Every time the channel to u comes up, v adds u to formingv and sends its height in an Update message to u. When u gets the message from v, it updates the entry for v in its height array, but does not send its own height back to v. As long as u's height does not change, u does not send its height to v. Thus v is never able to move u from formingv into Nv.

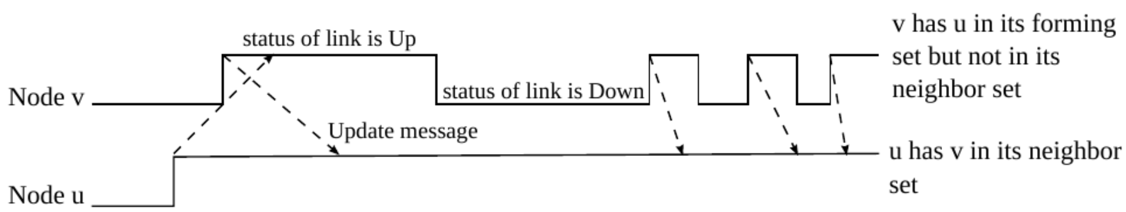

Figure 4.1: The status of the channel from u to v remains Up, but the status of the channel from v to u fluctuates.

However, we are assured by Lemma 1 below that after time tLTC , $N_u \cup forming_u$ does not change for any node u. Furthermore, a node u always sends Update messages to all nodes in $N_u \cup forming_u$, which constitutes all the outgoing channels of u.

Lemma 1 After time t_{LTC} , $N_u \cup forming_u$ does not change for any node u.

Proof When ChannelDownuv occurs, u removes v from both its Nu and formingu variables. When ChannelUpuv occurs, u adds v to its formingu variable and sends an Update message to v. When u receives an Update message from a node v, the only possible change to the Nu and formingu variables is that v is moved from formingu to Nu , which does not change $N_u \cup forming_u$.

tT LC is the latest among all the times at which either a ChannelDown, or a Chan- nelUp occurs. After this time, the only change to the N set or the forming set must be due to receipt of an Update message, causing lines 2 and 3 of Figure 2 to be executed. Thus the only change to the N set or the forming set is that a node which is removed from the forming set is added to the N set. This does not affect $N \cup forming$.

4.2 Height Tokens and Their Properties

Since a node makes algorithm decisions based solely on comparisons of its neighboring nodes' height tuples, we first present several important properties of the tuple contents. Define h to be a height token for node u in a configuration if h is in an Update message in transit from u, or h is the entry for u in the height array of any node. Let LP(h) be the leader pair of h, RL(h) the reference level (triple) of h, $\delta(h)$ the δ value of h, lts(h) the absolute value of the (nonpositive) leader timestamp (component nlts) of h, and $\tau(h)$ the τ value of h.

Given a configuration in which Channel(u, v) has status Up and $u \in N_v$, the (u, v) height sequence is defined as the sequence of height tokens $h0, h1, ..., hm$, where h0 is u's height, hm is v's view of u's height, and $h1, ..., h_{m-1}$ is the sequence of height to- kens in the Update messages in transit from u to v. If the status of Channel(u, v) is Up but $u \notin N_v$, then the (u, v) height sequence is defined similarly except that $h1, ..., h_m$ is the sequence of height tokens in the Update messages in transit from u to v; in these cases, v does not have an entry for u in its height array. If Channel(u, v) is Down, the (u, v) height sequence is undefined.

Property A If h is a height token for a node u in the (u, v) height sequence, then:

1. $lts(h) \leq \mathcal{T}_u$ and $\tau(h) \leq \mathcal{T}_u$
2. If h is in v's height array then $lts(h) \leq \mathcal{T}_v$ and $\tau(h) \leq \mathcal{T}_v$.

Proof By induction on the configurations in the execution.

Basis: In the initial configuration C_0 , all the leader timestamps and τ values are 0 and $\mathcal{T} \geq 0$ for all nodes v.

Inductive Hypothesis: Suppose the property is true in configuration Ci-1 and show it remains true in configuration Ci . Since the property is true in C_{i-1} , for every height token h in the (u, v) height sequence, we have:

(i) $lts(h) \leq \mathcal{T}_u(Ci - 1)$ and $\tau(h) \leq \mathcal{T}_u(C_{i-1})$

(ii) If h is in v's height array then lts(h) $\leq \mathcal{T}_v(C_{i-1})$ and τ (h) $\leq \mathcal{T}_v(C_{i-1})$

Inductive Step: If h is a preexisting height token during event ei (the event immediately preceding C_i), then by the inductive hypothesis and the increasing property of Tu , it follows that $lts(h) \leq \mathcal{T}_u(C_i)$ and $\tau(h) \leq Tu(Ci)$. If, on the other hand, h is created during event ei , then any new values of lts and τ generated by u are equal to Tu (Ci) and, thus, the property remains true.

If h is a height token for node u at some other node v, then h was either present at v during C_{i-1} or was received at v during event ei , immediately preceding Ci . In the first case, by the inductive hypothesis and the increasing property of Tv , it follows that $lts(h) \leq Tv(Ci)$ and $\tau(h) \leq Tv(C_i)$. In the second case, there exists a message through which v received h from u during event ei . Since T preserves causality, by the definition of the happens before relation, it follows that the creation of either $\tau(h)$ or $lts(h)$ preceded the receipt of the message by v. Therefore, in configuration Ci it remains true that $lts(h) \leq Tv(Ci)$ and $\tau(h) \leq Tv(Ci)$.

Property B, given below, states some important facts about height sequences. If the channel's status is Up and m = 1, meaning that no messages are in transit from u to v, then Part (1) of Property B indicates that v has an accurate view of u's height. If there are Update messages in transit, then the most recent one sent has accurate information. Part (2) of Property B implies that leader pairs are taken on in decreasing order. Part (3) of Property B implies that reference levels are taken on in increasing order with respect to the same leader pair. Note that Property B only holds if m ¿ 0.

Property B: Let h0 , h1 , . . . , hm be the (u, v) height sequence for any Channel(u, v) whose status is Up. Then the following are true if m ¿ 0:

1. $h_0 = h_1$.
2. For all l, $0 \leq l < m, LP(hl) \leq LP(h_l + 1)$.
3. For all l, $0 \leq l < m, if LP(hl) = LP(hl + 1)$, then $RL(hl) \geq RL(hl + 1)$.

Proof The proof is by induction on the execution.

Initially in C0 , Channel(u, v) is either Up or Down. If Channel(u, v) is Down, then the (u, v) height sequence is undefined. If Channel(u, v) is Up, then the definition of initial configurations states that no messages are in transit and v has an accurate view of u's height, that is, m = 1 and h0 = h1 .

Suppose the property is true in configuration Ci-1 and show it is still true in configuration Ci .

Suppose event ei is ChannelDownuv. Then the (u, v) height sequence is not defined in Ci .

Suppose event ei is ChannelUpuv. By the assumption that the channel up/down events for a given channel alternate, the state of the channel in Ci-1 is Down and there are no messages in transit. Thus in Ci the (u, v) height sequence is h, h, where h is the height of u in Ci , which is stored in u's height array and is in the Update message that u sends to v. Clearly this height sequence satisfies the three conditions.

Suppose event ei is the receipt by v of an Update message from u. In one case, the (u, v) height sequence changes by dropping the last element, if the oldest message in transit takes the place of v's view of u's height. In the other case, the (u, v) height sequence does not change if the receipt causes v to record u's height and add u to Nv . In both cases, the three conditions still hold.

Suppose event ei is the receipt by u of an Update message from node w or is a ChannelDown event for a channel to some node other than v. If u does not change its height, then there is no change affecting the property.

Suppose u changes its height from h'0 to h.

Let the (u, v) height sequence in Ci-1 be h'0 , h'1 , . . . , h'm . By the inductive hypothesis, h'0 = h'1 . By the code, the (u, v) height sequence in Ci is h, h, h'1 , . . . , h'm . In each case we just have to show that h has the proper relationship to h'1 , which equals h'0 .

Case 1: ei calls $REFLECTREFLEVEL$: All of u's neighbors are viewed as having the same LP as u, having reference level (t, p, 0) for some t and p, and having a larger height than u.

Since u is a sink during the step, RL(h'0) $\leq$ (t, p, 0). Since RL(h) = (t, p, 1), and the old and new LP are the same, the property holds.

Case 2: ei calls ELECT S ELF: By Property A, lts in LP(h'0) is less than or equal to Tu' in configuration Ci-1 . The new leader pair has lts Tu in configuration Ci , which is greater than Tu' . So LP(h) $\leq$ LP(h'0).

Case 3: ei calls START N EW R EF L EVEL: By Property A, the τ value in RL(h'0) is less than or equal to Tu' at configuration Ci-1 . The new reference level has τ value Tu at configuration Ci , which is greater than Tu' and the LP is unchanged. So LP(h) = LP(h'0) and RL(h) $\geq$ RL(h'0).

Case 4: ei calls PROPAGATE L ARGEST R EF L EVEL: All neighbors of u are viewed as having the same LP as u, but with different RL's among themselves, and as having larger heights than u. By the code, u takes on the largest neighboring RL, which is at Case 4: ei calls PROPAGATE L ARGEST R EF L EVEL: All neighbors of u are viewed as having the same LP as u, but with different RL's among themselves, and as having larger heights than u. By the code, u takes on the largest neighboring RL, which is at least as large as u's old RL, since u is a sink. The LP is unchanged. So LP(h) = LP(h'0) and RL(h) $\geq$ RL(h'0).

Case 5: ei calls ADOPT LPI F P RIORITY: By the code, the new LP is smaller than the previous, so LP(h) ¡ LP(h'0).

4.3 Bounding the Number of Elections

In this subsection, we show that every node elects itself at most a finite number of times after the last topology change.

Define the following with respect to any configuration in the execution. For LP (-s, l), where Tl (t) = s and t $\geq$ tLTC , let LP tree LT (-s, l) be the sub-graph of the connected component whose vertices consist of all nodes that have taken on LP (-s, l) in the execution (even if they no longer have that LP), and whose directed edges are all ordered pairs (u, v) such that v adopts LP (-s, l) due to the receipt of an Update message from u. Since a node can take on a particular LP only once by Property B, LT (-s, l) is a tree rooted at l.

Porperty C: For each height token h with RL (t, p, r), either t = p = r = 0, or t ¿ 0, p is a node id, and r is 0 or 1.

Poof The proof is by induction on the sequence of configurations in the execution. The basis follows since all height tokens in an initial configuration have RL (0, 0, 0).

For the inductive step, we consider all the ways that a new RL can be generated (as opposed to copying an existing one). In ELECT S ELF, the new RL is (0,0,0). In START N EW R EF L EVEL , the new RL is (t, p, 0), where t is the current causal clock time, which is positive, and p is a node id. In $REFLECTREFLEVEL$, the new RL is (t, p, 1), where (t, p, 0) is a pre-existing height token. By the precondition for exe- cuting $REFLECTREFLEVEL$, t is positive. By the inductive hypothesis applied to the pre-existing height token (t, p, 0), p is a node id.

Property D: Let h be a height token for some node u. If $LP(h) = (-s, l)$, where for some global time t, $T_l(t) = s$ and $t \geq t_{LTC}$, then $RL(h) = (0, 0, 0)$ and $\delta(h)$ is the distance in $LT(-s, l)$ from l to u.

Proof: By induction on the sequence of configurations in the execution.

By Property A, the basis is configuration C j , just after the event at global time t when the first height tokens with LP (-s, l) are created. By the code, these height tokens are created by node l for itself and have RL (0, 0, 0) and $\delta = 0$.

Assume the property is true in configuration Ci-1 , with i - 1 $\geq$ j, and show it is true in configuration Ci . Since no further topology changes occur, the only possibility for event ei is the receipt of an Update message. Suppose node u receives Update(h) from node v.

As a result of the receipt of the message, u records h as v's height in its view. The inductive hypothesis implies that the property remains true for this new height token.

Also as a result of the receipt of the message, u might change its height.

Suppose u changes its height by executing ADOPT LPI F P RIORITY, adopting the LP in h, where LP(h) = (-s, l). By the inductive hypothesis, RL(h) = (0, 0, 0), and δ (h) is the distance from l to v in LT (-s, l) in C_{i-1} . By Property B, since u adopts (-s, l), it must be that u's LP is larger than (-s, l) in Ci-1 , and thus v is u's parent in LT (-s, l). By the code, u sets its RL to (0, 0, 0) and its δ to $\delta(h) + 1$. But this is exactly the distance in LT (-s, l) from l to u. So all height tokens created in this step satisfy the property.

Suppose u changes its height because it becomes a sink and u's new height has LP (-s, l). First, we show that u does not take on LP (-s, l) as a result of ELECT S ELF. By assumption, LP (-s, l) is created in configuration C j (the base case). By the code and the increasing property of causal clocks, it follows that l cannot create a duplicate of LP (-s, l) at some later configuration Ci . Therefore, u does not take on LP (-s, l) as a result of ELECT S ELF.

Thus, the old height of u, call it h' , also has LP (-s, l). Since u becomes a sink, all its neighbors have LP (-s, l) in u's view, and by the inductive hypothesis they all have RL (0, 0, 0) in u's view. Thus the new height of u is not the result of execut- ing $REFLECTREFLEVEL$ (which requires the neighbors' common τ to be positive) or PROPAGATE L ARGEST R EF L EVEL (which requires the neighbors to have different RL's). Instead, it must be the result of executing START N EW R EF L EVEL. Since u is a sink and (0, 0, 0) is the smallest possible RL by Property C, RL(h') = (0, 0, 0). Also, since u is a sink, u = l. Let v be u's parent in the LP-tree LT (-s, l) and let d be the distance in that tree from l to v. By the inductive hypothesis, in u's view of v's height, v's δ = d, but in u's own height, δ = d + 1. Thus the edge between u and v is directed toward v, and u cannot be a sink, a contradiction.

Lemma 2 Any node u that adopts leader pair (-s, l) for any l and any s, where for some global time t, Tl (t) = s and t ¿ tLTC , never subsequently becomes a sink.

Proof Suppose in contradiction that u adopts leader pair (-s, l) at global time t1 ¿ t and that at global time t2 ¿ t1 , u becomes a sink. Suppose u does not change its leader pair in the time interval (t1 ,t2). (If u did change its leader pair, the new leader pairs would all be smaller than (-s, l) by Property B, and the argument would still hold with respect to the latest leader pair taken on by u in that time interval.)

Let v be the parent of u in the LP-tree LT (-s, l). Immediately after time t1 , the link (u, v) is directed from u to v in u's view.

In order for u to become a sink at time t2 , there must be some time between t1 and t2 when the link (u, v) reverses direction in u's view. Suppose the link reverses because u's height lowers. Recall that u does not change its leader pair in (t1 ,t2) by assumption. By Property D, u's reference level remains (0, 0, 0) in (t1 ,t2) and u's δ stays the same in the interval. That is, u's height does not change, and in particular does not lower. Thus the only way that the link (u, v) can reverse direction in (t1 ,t2) is due to the receipt by u of an update message from v with a new height for v that is higher than u's height.

How can v's height change after v takes on leader pair (-s, l)? One possibility is that v's leader pair changes. By Property B, any change in v's leader pair will be to a smaller one, which will be adopted by u together with a δ value that keeps the link directed from u to v in u's view.

The other possibility is that v's leader pair does not change but some other component of its height changes. But by Property D, since v's leader pair has timestamp -s with Tl (t) = s and t ¿ tLTC , v's RL and δ cannot change.

Thus no change to v's height reported to u after time t1 can cause the link (u, v) to be directed from v to u in u's view, and u cannot be a sink at time t2 , which is a contradiction.

Lemma 3 No node elects itself more than a finite number of times after global time tLTC.

Proof Suppose in contradiction that a node u elects itself an infinite number of times after the last topology change. Once it has elected itself the first time, the only way it can become a sink and elect itself again is by adopting a new LP first. Thus, node u needs to adopt new LP's infinitely often after tLTC . By Property B, the leader times- tamp of each subsequent LP has to be greater than the previous one, which results in an increasing sequence of leader timestamps that u adopts. Let Tmax be the maximum of the clocks of all nodes at time tLTC . In the process of adopting increasing leader timestamps, at some point u will adopt LP(-s, l) where Tl (t) = s and for which s ¿ Tmax .

This follows from the first property of causal clocks which states that for each node u, the values of Tu are increasing, i.e., if ei and e j are events involving u in the execution with i ¡ j, then Tu (ei) ¡ Tu (e j), and, furthermore, if there is an infinite number of events involving u, then Tu increases without bound.

Because Tmax was the maximum value of all clocks at the time of the last topology change, it follows that t ¿ tLTC . By Lemma 2, however, node u does not become a sink after it has adopted LP(-s, l) and thus it cannot elect itself again after that time, which is a contradiction.

If we use perfect clocks to implement T , we can get a stronger bound on the number of times a node elects itself after the last topology change. In fact, with perfect clocks it is guaranteed that no node elects itself more than once after the last topology change, as we now explain. As stated in the proof of Lemma 3, if a node u elects itself more than once after the last topology change, it must take on a new LP in between each successive pair of elections. Also, by Property B, the timestamps in these LP's must be increasing. As explained in the proof of Lemma 3, there could be multiple LPs already existing at the time of the last topology change whose timestamps are greater than the timestamp of the LP that u takes on the first time it elects itself after the last topology change. The reason is that the clocks are causal, yet are drawn from a totally-ordered set, and thus just because clock value t1 is less than clock value t2 , it does not follow that the event associated with t1 happened before the event associated with clock value t2 . However, the number of such misleading timestamps is finite, so eventually, if u keeps electing itself, it will take on a timestamp that is associated with an event that occurred after the last topology change. Then we can apply Lemma 2 to deduce that u will never elect itself again. When clocks are perfect, however, there can be no such misleading timestamps in LP's: if the timestamp in a new LP is greater than the timestamp taken on by u the first time, then this LP was definitely generated after the last topology change and Lemma 2 applies immediately. For more details, refer to Lemma 3 in [15].

4.4 Bounding the Number of New Reference Levels

In this subsection, we show that every node starts a new reference level at most a finite number of times after the last topology change. The key is to show that after topology changes cease, nodes will not continue executing Line 13 of Figure 2 infinitely and will therefore stop sending algorithm messages. First we show that the δ value of a node does not change unless its RL or LP changes.

Property E: If h and h' are two height tokens for the same node u with RL(h) = RL(h') and LP(h) = LP(h'), then δ (h) = δ (h').

Proof Initially, in C0 , the only height tokens for node u are the ones in u and the ones in u's neighbors, and the neighbors have accurate views of u's height.

Suppose the property is true through configuration Ci-1 . We will show it is still true in the next configuration Ci . The only way that new height tokens can be intro- duced into the system is if a node u changes its height and sends Update messages with the new height to its neighbors.

Suppose u changes its height through ELECT S ELF (resp., START N EW R EF L EVEL). Since the new height's leader timestamp (resp., τ) is the value of the logical clock of u, Property A implies that there is no pre-existing height token for u in the system with the new leader timestamp (resp., τ). Thus there cannot be two height tokens for u with the same RL and LP but conflicting δ s.

Suppose u changes its height through ADOPT LPI F P RIORITY. Then the new height of u has a smaller LP than the old height. By Property B, there is no pre-existing height token for u in the system with the new LP. Thus there cannot be two height tokens for u with the same RL and LP but conflicting deltas.

Suppose u changes its height through $REFLECTREFLEVEL$. Since u is a sink and in its view all its neighbors have a common, unreflected, RL, call it (t, p, 0), u's RL must be at most (t, p, 0). Since u's new RL is (t, p, 1), Property B implies that there is no pre-existing height token for u in the system with the new RL. Thus there cannot be two height tokens for u with the same RL and LP but conflicting δ s.

Suppose u changes its height through PROPAGATE L ARGEST R EF L EVEL. The pre- condition includes the requirement that not all the neighbors have the same RL (in u's view). Since u becomes a sink, u's old RL is less than the largest RL of its neighbors, which is the RL that u takes on in Ci . Property B implies that there is no pre-existing height token for u in the system with the new RL.

Thus there cannot be two height tokens for u with the same RL and LP but con- flicting δ s.

The next definition and its related properties are key to understanding how un- reflected and reflected reference levels spread throughout the connected component after the last topology change.

Define the following with respect to any configuration in the execution after tLTC . For global time t ' $\geq$ tLTC , let the RL DAG RD(t, p), where T p (t ') = t, be the sub- graph of the connected component whose vertices consist of p and all nodes that have taken on RL prefix (t, p) by executing either PROPAGATE L ARGEST R EF L EVEL or $REFLECTREFLEVEL$ in the execution (even if they no longer have that RL pre- fix). In RD(t, p), the directed edges are all ordered pairs of node ids (u, v) such that u $\in$ Nv and v $\in$ Nu and u has RL prefix (t, p) prior to the event in which v first takes on RL prefix (t, p). We say that node u is a predecessor of node v in RD(t, p) and v is a successor of u in RD(t, p).

Property F: If there is a height token for node u with RL prefix (t, p), where T p (t ') = t and t ' $\geq$ tLTC , then u is in RD(t, p).

Proof By induction on the sequence of configurations in the execution.

The basis is configuration C j , where gt(C j) = t ' , i.e., the time when node p starts RL (t, p, 0). By Property A, there is no height token with RL prefix (t, p) in C j-1 , so the only height tokens we have to consider are those created by p, for p. By definition, p is in RD(t, p).

Suppose the property is true through configuration Ci-1 . We will show it is true in Ci .

Suppose in contradiction, in event ei , some node u takes on RL prefix (t, p) by calling ADOPT LPI F P RIORITY after receiving an update message from neighbor v containing height h with RL prefix (t, p). By the inductive hypothesis, v is in RD(t, p).

Let (-s, l) be LP(h). We are going to show that when v takes on RL prefix (t, p), f inal it already has LP (-s, l). We know that v must have a path to node p in Gchan that ' has been in place since p started the new RL prefix at time t , by the assumption that topology changes have stopped by real time t τ . Just before time t τ , all the neighbors of p had LP (-s, l) and RL prefix lower than (t, p), by Property B, or p would not have started a new reference level for LP (-s, l). Since the neighbors of p had LP (-s, l), they would have sent messages containing that LP to their neighbors prior to time t τ . Likewise, those neighbors would have messages in transit to their neighbors containing the LP (-s, l) and so on. In short, if the LP (-s, l) is adopted by any nodes that have a path to p at t τ , then the LP would have been adopted when that LP spread through the network with a lower RL prefix.

Thus, when v puts h in transit to u, there is already ahead of it in the (v, u) height sequence a height token for v's old height, with LP (-s, l). Since the channels are FIFO and no messages are lost after time t τ , u has already received the old height from v before ei . So in Ci-1 , u has a LP that is (-s, l) or smaller already, before handling the Update message with height h. Thus u does not execute ADOPT LPI F P RIORITY in ei , contradiction.

Property G: If there is a height token for node u with RL (t, p, 1), where for some global time t ' , T p (t ') = t and t ' $\geq$ tLTC , then all neighbors of u are in RD(t, p).

Proof By induction on the sequence of configurations in the execution.

The basis is the configuration C j with gt(C j) = t ' , i.e., the time when the new RL is started at node p. By Property A, there is no height token in C j-1 with RL (t, p, 1), and in C j we only add height tokens for node p with RL (t, p, 0). So the property is vacuously true.

Suppose the property is true through configuration Ci-1 and show it is true in Ci , i ¿ j.

By Property F and the definition of RD(t, p), the only way that u can take on RL (t, p, 1) is by $REFLECTREFLEVEL$ or PROPAGATE L ARGEST R EF L EVEL.

Suppose u takes on RL (t, p, 1) due to $REFLECTREFLEVEL$. Then all u's neigh- bors have RL (t, p, 0) in its view. By Property F, then, they are all in RD(t, p).

Suppose u takes on RL (t, p, 1) due to PROPAGATE L ARGEST R EF L EVEL. Thus there is a height token in Ci-1 for some neighbor v of u with RL (t, p, 1). By the inductive hypothesis applied to v, all of v's neighbors, including u, are in RD(t, p). Thus u's RL prefix at some earlier time is (t, p). By Property B (since the LP does not change in this interval), u's RL prefix in Ci-1 is at least (t, p). Since u is a sink during event ei , u's RL prefix in Ci-1 is at most (t, p), so it is exactly (t, p) in Ci-1 . Since u is a sink, every neighbor of u (in u's view) has RL prefix at least (t, p), and since (t, p, 1) is the maximum of the neighboring RL's, every neighbor of u (in u's view) has RL prefix exactly (t, p). Thus by Property F, every neighbor of u is in RD(t, p).

Property H: Suppose that u and v are two nodes such that u $\in$ Nv and v $\in$ Nu after tLTC . Consider two height tokens, hu for node u with RL(hu) = (t, p, ru) and δ (hu) = du , and hv for node v with RL(hv) = (t, p, rv) and δ (hv) = dv , where T p (t ') = t and t ' $\geq$ tLTC . Then the following are true:

1. If ru ¡ rv , then u is a predecessor of v in RD(t, p). If u is a predecessor of v in RD(t, p) then ru $\leq$ rv .
2. If ru = rv = 0, then du ¿ dv if and only if u is a predecessor of v.
3. If ru = rv = 1, then dv ¿ du if and only if u is a predecessor of v.

Proof By induction on the sequence of configurations in the execution.

Basis: Consider configuration C j , where gt(C j) = t ' , that is, when node p starts the new reference level (t, p, 0). By Property A, in configuration C j-1 , there are no height tokens with RL prefix (t, p). The only new height tokens introduced by event e j are those for p with RL (t, p, 0), and the RL DAG RD(t, p) consists solely of node p. Thus all parts of the property are vacuously true.

Induction: Assume the property holds through configuration Ci-1 and show it is true in Ci , i ¿ j.

By Property E, it is sufficient to consider the height tokens in u's view, since there cannot be other height tokens with the same RL and LP but different δ s.

Suppose new height tokens with RL prefix (t, p) are created by node u during event ei . The only ways this can happen are via REFLECT R EF L EVEL and PROPA - GATE L ARGEST R EF L EVEL , by Property F.

C ASE 1: REFLECT R EF L EVEL. During the execution of ei , all of u's neighbors are viewed by u as having RL (t, p, 0) and the new height tokens created for u have RL (t, p, 1).

We now show that u's RL prefix is less than (t, p) in Ci-1 . Suppose in contradiction u has RL (t, p, 0) in Ci-1 . By the inductive hypothesis, part (2), u's δ value cannot be the same as that of any of its neighbors. This is true since u and all its neighbors are in RD(t, p) by Property F, and, for any pair of neighboring nodes in RD(t, p), one is the predecessor of the other, since two events cannot happen simultaneously. Since u is a sink, its δ value must be smaller than those of all its neighbors. By the inductive hypothesis, part (2), u is a successor of all its neighbors, of which there is at least one.

Then at some previous time t " ¡ gt(Ci-1), u executed PROPAGATE L ARGEST R E - F L EVEL and took on RL (t, p, 0). This must be how u took on (t, p, 0) since, by Property F, u cannot take on RL (t, p, 0) by running ADOPT LPI F P RIORITY, and, if u = p, u has no predecessors in RD(t, p), contradicting the deduction that u is a suc- cessor of at least one neighbor. At t " , u has (in its view) at least one neighbor with RL (t, p, 0), (t, p, 0) is the maximum RL of all u's neighbors, and at least one neighbor, say v, has a smaller RL than (t, p, 0), albeit larger than u's (since u is a sink).

Suppose u has height hu at time t " , and its view of v's height is hv at time t " . Since u is a sink, hu and hv have the same leader pair, say l p1 , we have

$$RL(hu) < RL(hv) < (t, p, 0) \tag{4.1}$$

This means that there was a previous time t "' ¡ t " when v actually took on height hv (with leader pair l p1). We also know that v has taken on (t, p, 0) before time t " , since u is a successor of all its neighbors and it takes on RL (t, p, 0) at time t " . Note that v could not have taken on RL (t, p, 0), with leader pair l p1 before t "' . This is because at t "' its leader pair is also l p1 and its height RL(hv) ¡ (t, p, 0). By Property B two height tokens with the same leader pair must have increasing reference levels. Hence, v took on (t, p, 0) after t "' and before t " . Suppose v took on (t, p, 0) at time s such that t "' ¡ s ¡ t " . We know that v has to be a sink at time s in order to do so. Thus at time s all v's neighbors in v's view have the same leader pair as itself, and v takes on (t, p, 0) with leader pair l p1 either by PROPAGATE L ARGEST R EF L EVEL or START N EW R EF L EVEL . Suppose v's own height is h'v at time s and its view of u's height is h'u . Both h'v and h'u have leader pair l p1 and, since v is a sink we have

$$h'v < h'u \tag{4.2}$$

Note that hv , hu , h'v , and h'u all have leader pair l p1 . We also know that hu ¡ hv from (1). Now from Property B

$$hv \leq h'v \tag{4.3}$$

Also from Property B

$$hv \leq h'v \tag{4.4}$$

Hence, from (1), (3) and (4), we have

$$h'u \leq hu < hv \leq h'v \tag{4.5}$$

This is in contradiction to (2). Part (1): All neighbors of u are its predecessors in RD(t, p) and in Ci , the prede- cessors of u have r = 0 and u has r = 1 so this part continues to hold. Part (2): The creation of the new height tokens does not affect this part, since the new tokens do not have r = 0. Part (3): Since u is not in RD(t, p) in Ci-1 , Property G implies that there cannot be a height token for any of u's neighbors with RL (t, p, 1), and this part is vacuously true. C ASE 2: PROPAGATE L ARGEST R EF L EVEL. In this case, u's neighbors have at least two different RLs so we need to consider which RL u propagates, (t, p, 0) or (t, p, 1). Case 2.1: Suppose u's new height has RL (t, p, 0). We first show that u has RL less than (t, p, 0) in Ci-1 . By the precondition for PROPAGATE L ARGEST R EF L EVEL, in u's view, (t, p, 0) is the largest neighboring RL, at least one neighbor has RL less than (t, p, 0), and u is a sink. Thus u's RL must be less than (t, p, 0). Part (1): Since the new height tokens of both u and its predecessors have reflection bit 0, this part is not invalidated in Ci . Part (2): Each of u's neighbors for which u has a height token h' with RL (t, p, 0) is a predecessor of u in RD(t, p), since u is not yet in RD(t, p). By the code, u's new height h has a δ calculated so that h' ¿ h. Part (3): The new height tokens do not have reflection bit 1 so this part is unaf- fected. Case 2.2: Suppose u's new height has RL (t, p, 1). Then the largest RL among u's neighbors has, in u's view, RL (t, p, 1). Property G implies that u is in RD(t, p). So the RL prefix of u is at least (t, p). Since u is a sink, its RL prefix is (t, p) in Ci-1 . So all neighbors (in u's view) have RL (t, p, 0) or (t, p, 1) and there is at least one neighbor with each RL. Consider any neighbor v of u with RL (t, p, 1) in u's view. By the inductive hy- pothesis, part (1), v must be a successor of u in Ci-1 . Consider any neighbor w of u with RL (t, p, 0) in u's view. By the inductive hypothesis, part (2), w must be a predecessor of u in Ci-1 . Part (1): Since u's new height causes it to have the same reflection bit as its suc- cessors, and a larger reflection bit than its predecessors, this part continues to hold in Ci . Part (2): Since the new height tokens do not have reflection bit 0, this part is not affected. Part (3): As argued above, each of u's neighbors v for which u has a height token h' with RL (t, p, 1) is a successor of u in RD(t, p). By the code, u's new height h has a δ calculated so that h' ¿ h.

Lemma 4 Every node starts a finite number of new RLs after tLTC .

Proof Suppose in contradiction that some node u starts an infinite number of new RLs after tLTC . Now we show that u takes on a new LP infinitely often. Suppose in contradiction that u does not do so. Let tLLP be the latest time at which u takes on a new LP. Consider the first and second times that u starts a new RL (for the same LP) after maxtLTC ,tLLP ; call these times t1 and t2 . At global time t1 , u sets its to 1 . Since u does not take on any more LPs, Property B implies that at the beginning of the step at time t2 , u's is at least 1 , which is positive. At the beginning of the event at time t2 , let (t, p, r) be u's RL and let (tc , pc , rc) be the common RL of all u's neighbors (in u's view). Thus the precondition for starting a new RL cannot be that tc = 0, otherwise u would not be a sink. So it must be that tc ¿ 0, rc = 1, and pc 6= u. There are two cases, depending on the relationship between (t, p) and (tc , pc) (note that (t, p) cannot be larger than (tc , pc) since u is a sink). Case 1: (t, p) ¡ (tc , pc). Since u has a height token with RL (tc , pc , 1) for each neighbor v, we can apply Property G to deduce that all neighbors of v, including u, are in RD(tc , pc). Thus, at some previous time, u has RL prefix (tc , pc). But Property B implies that it is not possible for u to have RL prefix (tc , pc) and then later to have RL prefix (t, p), since (t, p) ¡ (tc , pc). Case 2: (t, p) = (tc , pc). By Property F, node u is in RD(t, p). Thus u has a neigh- bor v that is a predecessor of u in RD(t, p). Here we know that v is in Nu . Also, since v is a predecessor of u in RD(t, p) u is in Nv . Hence, we can apply Property H. Since in u's view, v has RL (t, p, 1), Property H, Part (1), implies that u's reflection bit must also be 1, and Property H, Part (3), implies that u's height must be greater than v's. But this contradicts u being a sink. Since u takes on a new LP infinitely often, by Property B, the lts values of the LP's that u adopts are increasing without bound. Let Tmax be the maximum of the clocks of all nodes at time tLTC . Since u is adopting LPs with bigger leader timestamps, at some point in time it will adopt LP(s,) where for some global time t, T (t) = s and for which s ¿ Tmax . Because Tmax is the maximum of all clocks at the time of the last topology change, we can conclude that t ¿ tLTC . But then by Lemma 2, u is never again a sink after that time, contradicting the assumption that u starts a new RL infinitely often.

4.5 Bounding the Number of Messages

In this subsection we show that eventually no algorithm messages are in transit. f inal Lemma 5 Eventually all nodes in the same connected component of graph Gchan have the same leader pair. f inal Proof Choose a connected component of Gchan . Lemma 3 implies that there are a finite number of elections. Thus there is some smallest LP that ever appears in the connected component at or after tLTC , say (s,). Suppose in contradiction, it is not f inal true that eventually all nodes in the same connected component of Gchan have the same leader pair. We know that causal clocks have the property that for each node u, the values of Tu are increasing (i.e., if ei and e j are events involving u in the execution with i ¡ j, then Tu (ei) ¡ Tu (e j)), and, furthermore, if there is an infinite number of events involving u, then Tu increases without bound. We also know from Lemma 3 that no node elects itself more than a finite number of times after global time tLTC . From this and from Property B we know that eventually every node in the connected component will stop changing its leader pair. We can then partition the connected component into two sets of nodes, those that have adopted (s,) and those that have f inal not. Thus there exist two nodes u and v such that there is an edge in Gchan between u and v, and u's final leader pair is (s,), whereas v's final leader pair is not (s,). Case 1: If (s,) originated at or after tLTC then both communication channels f inal (from u to v and v to u) exist in Gchan . Suppose the last ChannelUpuv event occurs at time t tLTC . After time t, v is in formingu and, by the code, v is not removed from formingu , since no ChannelDownuv event occurs after this time. By Lemma 1 there is no change in Nu formingu after tLTC , hence v is either in Nu or formingu after tLTC . In either case, when u adopts (s,), v gets an Update from u and adopts (s,). This leads to a contradiction. Case 2: Suppose (s,) originated before tLTC . We know that there is a last Chan- nelUp event at u for v (since the channel is eventually Up after tLTC). Suppose this ChannelUp event occurs at time t. If at time t node u has already taken on leader pair (s,), then u will send an Update message to v with (s,). If node u takes on leader pair (s,) at time t ¿ t, then u will send an Update message to v with (s,) at time t . In either case node v will receive this Update message. Since node v does not take on leader pair (s,), it must be that v ignores this message, because the Channelvu is down and u is neither in formingv nor in Nv . However, in this case there will be at time t ¿ t , a last ChannelUp event at v for u (since the channel is eventu- ally U p after tLTC). At time t v will send its height h (with a leader pair older than (s,)) to u. At this time node u detects that v has an older leader pair (since v has not taken on (s,)) and node u sends an Update message with (s,) to v. When v receives this message with a more recent leader pair (s,), v adopts this leader pair. This is a contradiction to the assumption that u and v have different leader pairs. Lemma 6 Eventually there are no messages in transit. Proof By Lemma 5, eventually every node in the connected component has the same LP, say (s,). Lemma 4 states that there are a finite number of new RLs started. Thus there is a maximum RL that appears in the connected component associated with the common LP (s,). Let t be some global time after the last RL has been started and the last leader has been elected. Assume in contradiction that messages are always in transit. Since every message sent is eventually received, there must be an infinite number of Update messages sent. Thus, infinitely often after time t, an Update message is received that causes the recipient to (temporarily) become a sink, change its height, and send new Update messages. Since there are no more elections or new RLs started after time t, the ac- tions taken by the recipients are REFLECT R EF L EVEL and PROPAGATE L ARGEST R E - F L EVEL . Thus eventually every node has the same, maximum, RL. Once all nodes have the same RL, the only possible action when a node becomes a sink is to run ELECT S ELF or START N EW R EF L EVEL . But this contradicts the fact that after time t these events do not happen. The previous lemma, together with Property B, gives us this corollary: Lemma 7 Eventually every node has an accurate view of its neighbors' heights. 4.6 Leader-Oriented DAG This subsection culminates in showing that eventually the algorithm terminates (i.e., no messages are in transit), with each connected component being leader-oriented. Property I: A node is never a sink in its own view. Proof By induction on the sequence of configurations in the execution. In the initial configuration, every node in every connected component is assumed to have RL (0,0,0), LP (, 0) where is a node in the same component, and a value such that it has a directed path to . Assume the property is true in configuration Ci1 and show it is true in Ci , i ¿ 0. Let u be the node taking the step ei . First consider the case when ei is the receipt of an Update message from a neigh- bor. If the neighbor's new height causes u to become a sink, then either u elects itself (in which case, by definition it is no longer a sink) or u reflects a reference level, starts a new reference level, or propagates a reference level. In each of the latter three cases, the code ensures that u is no longer a sink, as reflection manipulates the reflec- tion bit, starting a new reference level manipulates the component, and propagation manipulates the value appropriately. If the neighbor's new height causes u to adopt a new leader pair, then the code ensures that u is no longer a sink by manipulating the value appropriately (the new value is greater than that of the node which sent the Update message). If ei is a ChannelDown event, then any change to u's height through electing itself or starting a new reference level does not cause u to become a sink, as explained above. If ei is a ChannelUp event, then no change is made to any of the heights stored at u. Property J: Consider any height token h for node u. If RL(h) = (0, 0, 0), then (h) 0. Furthermore, (h) = 0 if and only if u is a leader. Proof By induction on the sequence of configurations in the execution. The basis follows by the definition of the initial configuration. Assume the property is true in configuration Ci1 and show it is true in Ci , i ¿ 0. Let u be the node taking the step ei . Suppose u elects itself. Then by the code, it sets its RL and to all zeroes, so the property holds. Now consider all the ways that u can change its RL and/or , other than by elect- ing itself. Reflection causes u to have a non-zero reflection bit, so the property holds vacuously. Starting a new reference level causes u to have a positive , so the property holds vacuously. Consider the situation when u propagates the largest reference level, say RL. The precondition for propagation is that u's neighbors have different reference levels, and thus RL must be larger than the reference level of another of u's neighbors. By Prop- erty C, then u's RL cannot be (0,0,0). Thus u's new height does not have reference level (0,0,0) and thus the property holds vacuously. Consider the situation when u adopts a new LP, because of the receipt of height h. If RL(h) = (0, 0, 0), then the inductive hypothesis shows that (h) 0, and thus u's new height has positive and the property holds. If RL(h) 6= (0, 0, 0), then the property holds vacuously. Theorem 1 Eventually the connected component is leader-oriented. Proof By Lemma 5, eventually all nodes in the component have the same LP, say (s,). By Lemma 7, every node eventually has an accurate view of its neighbors' heights. First, we show that node must be in the component. Suppose in contradiction that node is not in the component. Since cycles are not possible, there is some node in the component that has no outgoing links. But this node is not , since we are assuming is not in the component, and thus the node is a sink, violating Property I. Now that we know that node is in the component, we can proceed to show that the component is -oriented. Property J states that node , and only node , has RL (0,0,0) and zero . Property C implies no node has a negative number in its RL. Thus Property J implies that has the smallest height in the entire component and therefore has no outgoing links. Property I tells us that there are no sinks, so every node other than has an outgoing link. Since there are no cycles, the component is leader-oriented, where is the leader.

Chapter 5

Conclusion

We have described and proved correct a leader election algorithm using logical clocks for asynchronous dynamic networks. A set of circumstances were identified under which the algorithm does not elect a leader unnecessarily, but it remains to give a more complete characterization of such circumstances. Also, the time and message complexity of the algorithm needs to be analyzed. It would be interest- ing to compare the efficiency of the algorithm in the cases of perfect clocks and logical clocks.

Bibliography