

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

1	Introduction	1
2	Conclusion	3
	Bibliography	4

List of Figures

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 endpoints 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 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 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

Chapter 3

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 interesting to compare the efficiency of the algorithm in the cases of perfect clocks and logical clocks.

Bibliography