FAST AND MESSAGE-OPTIMAL SYNCHRONOUS ELECTION ALGORITHM FOR COMPLETE NETWORKS

Eli Gafni
Yehuda Afek
Leonard Kleinrock

October 1984
CSD-840041

FAST AND MESSAGE-OPTIMAL SYNCHRONOUS ELECTION ALGORITHM FOR COMPLETE NETWORKS

Eli Gafni
Yehuda Afek
Leonard Kleinrock

Computer Science Department
University of California, Los Angeles, CA 90024

1 Introduction

In the election problem, a single processor, the leader, has to be distinguished from a set of processors which differ only by their identifiers (*ids*). Initially, no processor is aware of the *id* of any other processor. The election algorithm is an identical program residing at each processor in the network. An arbitrary subset of processors spontaneously wake up at arbitrary times and start the algorithm by sending messages over the network. When the message exchange terminates, the leader is distinguished from all other processors.

This paper addresses the problem of electing a leader in a synchronous complete network. In such a network, each pair of processors is connected by a bidirectional communication link, and all processors are connected to a global clock. We assume that all the links incident to a processor on which no message was sent or received are indistinguishable.

For arbitrary asynchronous networks, a $\Theta(m+n\cdot\log n)$ bound on the message complexity was proved [3,4,5,7], where n and m are the total number of processors and links in the network. $\Omega(m)$ is a lower bound since no algorithm may terminate before sending at least one message over each link; otherwise, an untraversed link could be the only link connecting two parts of the network, each holding a separate election. Yet, Korach et al. [6] noted that the $\Omega(m)$ lower bound does not hold in complete networks, where an election algorithm may terminate after one processor has communicated with all its neighbors. Subsequently they proved a lower bound of $\Omega(n\cdot\log n)$ messages for the asynchronous complete network and presented an algorithm that requires $5n\cdot\log n+O(n)$ messages and $O(n\cdot\log n)$ time.

For synchronous complete networks, $\Omega(n\cdot\log n)$ is also a lower bound on the message complexity, as is recently shown in [1], where it is proved that $O(\log n)$ rounds is a

*This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-82-C-0064. The second author is an IBM Graduate Fellow.

lower bound on the time complexity of any message optimal synchronous algorithm.

In this paper we employ the observation of Korack et al. [6] and the fact that in a complete network every processor may be the root of a star spanning tree to construct a synchronous election algorithm for complete networks, which attains the above message and time lower bounds. The message complexity of the algorithm is $3n \cdot \log n$, and its time complexity is $2\log n$ rounds.

An $O(1)$ time and $O(n^2)$ messages algorithm is easily constructed as follows: every initiator sends messages to all its neighbors as soon as it wakes up. All the initiators will then elect the highest id initiator as the leader. The message complexity can be reduced to $O(n \cdot \log n)$ by using the algorithms suggested in [2] [1]. There, each initiator sends, at most, one message at a time, and thus the time complexity is increased to $O(n)$. In this paper we use the synchronous model of communication to achieve an $O(\log n)$ time complexity while maintaining the $O(n \cdot \log n)$ message complexity.

In Section 2 the model of a synchronous complete network is presented. Section 3 gives the algorithm. Throughout the paper, unless specified differently, log denotes $\log_2$.

2 Model and Assumptions

We model a synchronous complete network of n processors as follows: Each processor is connected by $n-1$ bidirectional communication links to all other processors. A single global clock is connected to all processors. The time interval between two consecutive pulses of the global clock is a *round*. In the beginning of each round, each processor decides, according to its state, on which links to send messages and what messages to send. Each processor then receives any message sent to it in this round and uses the received messages and its state to decide on its next state. All the links incident to a processor on which no message was sent or received are indistinguishable. Each processor starts its participation in the algorithm either by waking up spontaneously at the end of an arbitrary round, in which case it is called *initiator*, or by receiving a message of the algorithm.

3 Synchronous Election Algorithm

The algorithm is initiated by any subset of processors, each of which is a candidate for leadership. Each candidate tries to capture all other processors by sending messages on all the links incident to it. The candidate that has succeeded in capturing all its neighbors elects itself as the leader. To guarantee that only one processor succeeds, all candidates but one are *killed*.

1 - Although the mechanisms there are for asynchronous networks they can be easily modified to work on synchronous networks.

All candidates use a variable called *age* to count the number of rounds passed since they have started the algorithm. Every captured processor has an *owner–age* and *owner–id* variables which are the age and id of the oldest candidate from which it has received a message (age ties are resolved by selecting the highest id).

Every live candidate performs the following operation: In round $2i$, $i \geq 0$ of its algorithm, it sends 2^i messages containing its age and id to 2^i new neighbors in an attempt to capture them. If in round $2i+1$, the candidate receives acknowledgements to all the messages which it sent in round $2i$, it proceeds as a candidate to the next round. On the other hand, if not all the acknowledgements are received, the candidacy of the processor is eliminated. In every round, a processor acknowledges a message only if the $<age, id>$ pair of the message is lexicographically larger than that both of any other message it has received and of its owner. Whenever a processor acknowledges a candidate, that candidate becomes the processor's owner.

Figure 1 shows the formal description of the algorithm. Upon waking up, every processor in the network spawns a process, called *node*. Processors which wake up spontaneously spawn an additional process, called *candidate*. A logical bidirectional link, which behaves in the same way that a physical link does, is extended between the two processes of a spontaneously waking up processor. Thus, the candidate process of every initiator is connected by n links to n node processes, one in each processor in the network. Each candidate process tries to capture all the node processes in the network. Two types of messages are used by the algorithm, *candidate*, which includes the <age, id> pair of some candidate, and *acknowledgement*, which is the reply to a candidate message. All messages received by a processor are grouped according to their type. The candidate messages, which are sent by candidate processes, are forwarded to the node process. The acknowledgement type messages, which are generated by node processes, are forwarded to the candidate process.

Time and Message Complexities

Let p be the largest id candidate from the set of oldest candidates. We observe the following three facts:

Fact 1: The owner-age of every node strictly increases from round to round.

Fact 2 : $2\log n$ rounds after it was awakened spontaneously, candidate p has captured all the nodes and is elected as the leader of the network.

Fact 3 : At most, $\frac{n}{2^{i-1}}$ candidates reach age $2i$, $1 \leq i \leq \log n$.

```
/* The candidate process program */

  unused := { the set of $n$ links incident to the candidate process }
  age := -1 ;
  Each round do:
    age := age + 1 ;
    If age is even
    Then
      If unused is empty
      Then
        ELECTED, STOP
      Else
        $E$ := Minimum ( $2^{age/2}$, | unused | ) ;
        Send (age, id) over $E$ links from unused, and
        remove these links from the set unused ;
    Else   /*  age is odd  */
      Receive all acknowledgement type messages
      If received less than $E$ acknowledgements
      Then
        Stop          /*  Not a candidate any more */
  End each round.

/* The node process program */

  $L^*$ := nil ;
  owner-age := -1 ;
  Each round do:
    Send an acknowledgement over $L^*$ ;
    owner-age := owner-age + 1 ;
    Receive all candidate type messages { (age, id) over link L } ;
    Let ($age^*$, $id^*$) be the lexicographically largest  ( $age$, $id$) candidate message, and
    $L^*$ the link over which it arrived ;
    If ($age^*$, $id^*$)>($owner\text{-}age$, $owner\text{-}id$)
    Then
      ($owner\text{-}age$, $owner\text{-}id$ ):= ($age^*$, $id^*$) ;
    Else
      $L^*$ := nil ;
  End each round.
```

Figure 1: The Algorithm

Fact 1 follows immediately from the program for node processes. Fact 2 holds because all of p's candidate messages get acknowledged, and a node whose owner is p does not acknowledge any other message. Fact 3 follows fact 1 and the observation that every node acknowledges at most one message in which the age is i, $0 \leq i \leq \log n$, i.e., the sets of 2^{i-1} nodes that are captured by candidates at age $2i$ are disjoint.

According to fact 2, the time complexity of the algorithm is $2\log n$. Since every node sends at most 1 acknowledgement to candidate at age $2i$, the total number of acknowledgments is $n \cdot \log n$, each with $O(1)$ bits. Due to fact 3, the total number of candidate messages is $\sum_{i=1}^{\log n} \frac{n}{2^{i-1}} 2^i = 2n \cdot \log n$, each with $\log n + \log\log n$ bits.

A continuum of algorithms can be easily devised to close the gap between the trivial $O(1)$ time and $O(n^2)$ messages algorithm and the $O(\log n)$ time and $3n \cdot \log n$ messages algorithm. Each algorithm in the continuum is the same algorithm as the above, except for the fact that each candidate in age $2i$ is trying to capture c^i neighbors, $2 \leq c \leq n$. The time complexity of the algorithm is $2\log_c n$, and its message complexity is $2c \cdot n \cdot \log_c n$. In [1] it is proved that each of the algorithms in the continuum is optimal in the following sense: For a given c, any algorithm which is faster in the worst case will require more messages or, alternately, any algorithm which sends fewer messages is slower.

References

[1] Yehuda Afek and Eli Gafni, *Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks,* UCLA, Los Angeles, California. in preparation.

[2] Yehuda Afek and Eli Gafni, "Simple and Efficient Algorithms for Election in Complete Networks," in *Proceedings Twenty-Second Annual Allerton Conference on Communication, Control, and Computing,* Allerton, IL (October 3-5, 1984).

[3] J. E. Burns, "A Formal Model for Message Passing Systems," TR-91, Indiana Univ., Bloomington (May 1980).

[4] Greg N. Frederickson and Nancy A. Lynch, "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring," pp. 493-503 in *Proceedings of the 16th Ann. ACM Symp. on Theory of Computing,* Washington, D.C. (1984).

[5] Robert G. Gallager, Pierre A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans. Program. Lang. Syst.* **5**, pp.66-77 (Jan 1983).

[6] E. Korach, S. Moran, and S. Zaks, "Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors," in *Proceedings of the ACM Symp. on Principles of Distributed Computing,* Vancouver BC (August 1984).

[7] J. Pachl, E. Korach, and D. Rotem, "A Technique for Proving Lower Bounds for Distributed Maximum-Finding Algorithms," pp. 378-382 in *Proceedings of the 14th Ann. ACM Symp. on Theory of Computing,* San Francisco CA (1982).