TIME AND MESSAGE BOUNDS FOR ELECTION IN SYNCHRONOUS AND ASYNCHRONOUS NETWORKS

Yehuda Afek

February 1984
CSD-850003

TIME AND MESSAGE BOUNDS FOR ELECTION IN SYNCHRONOUS AND ASYNCHRONOUS COMPLETE NETWORKS
(Extended Abstract)

Yehuda Afek
Eli Gafni
Computer Science Department
University of California, Los Angeles, CA 90024

Abstract

This paper addresses the problem of distributively electing a leader in both synchronous and asynchronous complete networks. In the synchronous case, we prove a lower bound of $\Omega(n \cdot \log n)$ on the message complexity. We also prove that any message-optimal synchronous algorithm requires $\Omega(\log n)$ time. In proving these bounds we do not restrict the type of operations performed by nodes. The bounds thus apply for general algorithms and not just for comparison based algorithms. A simple algorithm which achieves these bounds is presented. In the asynchronous case we present a sequence of three simple and efficient algorithms, each an improvement on the previous, the last of which has time complexity $O(n)$ and message complexity $2 \cdot n \cdot \log n + O(n)$, thus improving the time complexity of the previously known best algorithm [10] by a factor of $\log n$.

1 Introduction

In the election problem, a single node, the leader, has to be distinguished from a set of nodes which differ only by their identifiers (*ids*). Initially no node is aware of the *id* of any other node. The election algorithm is an identical program residing at each node in the network. An arbitrary subset of nodes wake up spontaneously 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 nodes. This paper addresses the problem of electing a leader in a complete network. In such a network, each pair of nodes is connected by a bidirectional communication link. We consider both synchronous and asynchronous modes of communication.

For arbitrary asynchronous networks a $\Theta(m+n \cdot \log n)$ bound on the message complexity was proved [3,4,8,11], where n and m are the total number of nodes 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. [10] noted that the $\Omega(m)$ lower bound does not hold in complete networks, where an election algorithm may terminate after one node 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 networks, it was recently shown by Frederickson and Lynch [4] that one should distinguish between two types of algorithms: *general*, in which nodes may perform any computation on the values of their ids, and *comparison*, in which the values of ids can only be used for comparison with each other. They addressed the problem of election in a synchronous ring, for which they presented a general algorithm with $O(n)$ messages, and proved a lower bound of $\Omega(n \cdot \log n)$ messages for comparison algorithms, thus proving that general algorithms are strictly more powerful than comparison algorithms in a ring. This difference stems from the capability of general algorithms to delay messages and processors as a function of the value of their *id's*.

In this paper we prove that the message complexity of any election algorithm, comparison or general, in a complete synchronous or asynchronous network is $\Theta(n \cdot \log n)$, thus proving that general algorithms are no more powerful than comparison algorithms for the problem of election in complete networks. The difference between synchronous rings and synchronous complete networks stems from the fact that in a ring all nodes can be distributively awakened with n messages, whereas in the complete network the awakening problem is as hard as the election problem, hence requiring $\Omega(n \cdot \log n)$ messages. If all the nodes of a complete network could be awakened with n messages, then a general algorithm could take advantage of the synchronous mode of communication to elect a leader in a linear number of messages by using the principles suggested in [6].

We also prove an $\Omega(\log n)$ lower bound on the time complexity of any message optimal election algorithm in synchronous complete networks. Specifically, we show that if an algorithm, whether comparison or general, elects in at most $1/2 \log_c n$ rounds, then its message complexity is at least $\frac{c-1}{\log_c n} n \cdot \log n$.

Following the bounds, we present a synchronous algorithm which attains the above time-message tradeoff. When applying the algorithm in the asynchronous model its time degrades to $O(n)$ and its message complexity is $4n \cdot \log n$. The asynchronous algorithm is an improvement over the considerably more complicated algorithm in [10] which takes $O(n \cdot \log n)$ time and $5n \cdot \log n$ messages.

Finally, in an effort to reduce the message complexity of the asynchronous algorithm to $2n \cdot \log n$, we present a sequence of three algorithms (A, B, and C). The first two algorithms present a tradeoff between time and message complexities. Algorithm A (which was also derived independently in [9]) has an $O(n)$ time complexity and $2.773 \cdot n \cdot \log n$ message complexity. Algorithm B has an $O(n \cdot \log n)$ time complexity but a $2 \cdot n \cdot \log n$ message complexity. Analyzing the communication and time complexities of the two algorithms, we derive a third algorithm, algorithm C, whose time complexity is $O(n)$ and communication complexity is $2n \cdot \log n$, an improvement on the $O(n \cdot \log n)$ time and $2n \cdot \log n$ messages of [12]. It remains open whether a sublinear time, message optimal asynchronous algorithm exists. We conjecture that such an algorithm does not exist, i.e., that the time complexity of any asynchronous election algorithm is $\Omega(n)$.

In Section 2 we present the lower bounds for the synchronous case. Section 3 gives the synchronous algorithm which attains these bounds. Section 4 contains the asynchronous algorithms. Throughout the paper, unless specified differently we use log to denote $\log_2$.

2 Lower Bounds

2.1 Definitions and Assumptions

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

Let A be an election algorithm on a synchronous complete network. An *event* is the sending of a message over a previously unused link (Two messages, sent in the same round in opposite directions over a previously unused link, are considered two events). With each event we associate a pair (s,d), where s is the source node and d is the destination node of the corresponding message. With each round i of the algorithm we associate a set of events, R_i. A sequence $E=(R_0,R_1, \cdots)$ is called an *execution*. An *execution-prefix* E_j is a prefix, $(R_0,R_1, \ldots ,R_j)$, of an execution E. With each run of A we associate an execution, called a *legal-execution*, that includes all events which occurred in the run, arranged in the corresponding rounds. Henceforth, any mention of a message refers to an event.

A *cluster* in an execution-prefix E_j is a maximal subset of nodes spanned by a connected subnetwork of links which were used by events which occurred in E_j. The *degree* of a node v in an execution-prefix E_j is the number of links incident to v which were used by events in E_j. The *potential-degree* of node v in an execution-prefix E_j is the degree of v in E_j plus the number of times that v is a source node of an event in R_{j+1}. The *potential-degree* of a set of nodes is the largest potential-degree among its nodes.

For the purpose of proving the lower bound results we introduce a slightly different model, called the *stopping-model*. The stopping model allows us to withhold the clock pulse, in the beginning of round j, from a cluster of nodes (C) in E_{j-1}, given that no node in C is expected to receive a message, in round j from a node outside of C. The stopping-model will be used to avoid large differences in the clusters' growth rates. The nodes in C are then said to be *frozen* in round j. Therefore, a frozen node in a round neither sends nor receives any message in that round; nor does it change its state.

A *stopping-execution* is an execution which corresponds to a run of A in the stopping-model. A stopping-execution is a k *stopping-execution* if the cumulative number of pulses withheld over all clusters throughout the run, is k. Thus, a 0-stopping-execution is a legal-execution.

Lemma 1: For any k stopping-execution E there exists a k-1 stopping-execution E' which contains exactly the same events as E does.

sketch of proof: Let l be the minimum index round in which any cluster is frozen, and C a cluster which is frozen in l. An execution-prefix E' which satisfies the lemma can be obtained from E by shifting all events, which occurred before round l and involve nodes in C, one round forward. This effects, neither any event in later rounds, nor any event which involves nodes not in C. Notice that all nodes in C wake up in E' one round later than in E. ■

Corollary 2: For any stopping-execution there exists a legal-execution which contains the same events.

In the next two sections we will prove the lower bounds on the stopping model. By Corollary 2 these bounds apply to the correct model as well. In our proofs we do not restrict the type of operations performed by the nodes, hence proving the bounds for general algorithms.

2.2 Message Complexity Lower Bound

At the end of any election algorithm all nodes know who the leader is, hence any such algorithm has to send messages along the links of a spanning subnetwork, i.e., by the end of the algorithm the whole network is contained in one cluster. Thus, no cluster in the algorithm can defer indefinitely from sending messages to nodes not in the cluster, since the rest of the network might not wake up spontaneously. In the following proof of the lower bound we will use an adversary argument to construct a stopping-execution which contains at least $\frac{1}{2}n\log n$ events. In the beginning of each round, the adversary first determines which clusters to freeze and then determines the destination of messages sent in this round over previously unused links. The first feature is used to delay the formation of larger clusters until later rounds in the run, thus avoiding large differences in the clusters' growth rates; the second feature is used to send as many messages as possible within one cluster.

Theorem 3: For any election algorithm A on a synchronous complete network of n nodes there is a stopping execution which contains at least $\frac{n}{2}\cdot\log n$ events.

Corollary 4: The message complexity of any election algorithm in a synchronous complete network of n nodes is at least $\frac{n}{2}\cdot\log n$.

proof of Theorem 3: Assume that $n = 2^m$ (which can easily be relaxed). We define a sequence of partitions $(P_0, \ldots, P_m)$ on the nodes such that each subset in P_0 contains one node, and each subset in P_j contains two subsets from P_{j-1}, $1 \leq j < m$. Hence, each subset in P_j contains 2^j nodes.

We construct, in m steps, a sequence of stopping-execution-prefixes $(E_{i_0}, \ldots, E_{i_m})$, $i_0 = 0$, each being a prefix of the next. E_{i_0} is an empty execution-prefix in which all nodes have been awakened and the potential-degree of each is at least 1. This is possible by withholding the clock pulse from any node whose potential-degree is at least 1 until there is no node with potential-degree 0. Inductively we assume that: 1. Any cluster in E_{i_j} is contained within one subset in P_j. 2. The potential-degree, in E_{i_j} of every subset in P_j is at least 2^j. Obviously, E_{i_0} satisfies these assumptions.

Assuming that $E_{i_{j-1}}$ has been constructed, we describe how the adversary constructs E_{i_j}, $j<m$. In each round in the j^{th} step, we freeze all the subsets in P_j which contain at least one node whose potential-degree $\geq 2^j$. When all subsets are frozen step j is finished. The source and destination nodes of any message sent in this step are both in the same subset in P_j. This is always possible since every node that has a potential-degree $\geq 2^j$ is frozen. Clearly, E_{i_j} satisfies the inductive assumptions. In the m^{th} step no freezing takes place. After that step, the network is contained in one cluster and the algorithm is assumed to produce no more events.

Clearly, there are at least $^n/_{2^j}$ nodes whose potential-degree is at least 2^j, for $j=0, \ldots, m-1$. Thus, the total number of events is at least $^n/_2 \cdot \log n$. ■

Given that the message complexity of any election algorithm on a synchronous complete network is $\Omega(n \cdot \log n)$, the question arises how fast a message optimal algorithm can be. In the next section we prove that the time complexity of any message optimal algorithm is $\Omega(\log n)$.

2.3 Time Complexity Lower Bound

In this section we will extend the techniques of the previous one to prove, that the shorter the execution length, the larger the lower bound to the number of events it must contain.

Theorem 5: Any stopping-execution of an election algorithm in a synchronous complete network of n nodes which terminates in less than $^1/_2 \log_c n$ rounds, contains at least $\frac{c-1}{\log c} \cdot n \cdot \log n$ events.

Corollary 6: The time complexity of any message optimal election algorithm in a synchronous complete network of n nodes is $\Omega(\log n)$ rounds.

proof of Theorem 5: Let A be an election algorithm whose time complexity is at most $^1/_2 \log_c n$. Assume that $n=c^m$ (which can easily be relaxed). A construction similar to the one in the proof of theorem 3 will be used here. We construct, in m steps, a sequence of stopping-execution-prefixes $(E_{i_0}, \ldots, E_{i_m})$, $i_0=0$, each being a prefix of the next, and a sequence of partitions $(P_0, \ldots, P_j)$, the subset of each containing c subsets of the previous. E_{i_0} is an empty execution-prefix in which all nodes are awakened spontaneously. In P_0 each subset contains one node, thus, each subset in P_j contains c^j nodes. Inductively we assume that: 1. Any cluster in E_{i_j} is contained within one subset of P_j, and 2. The potential-degree, in E_{i_j} of every subset in P_j is at least c^j. Obviously, both E_{i_0} and P_0 satisfy these assumptions.

Assuming that $E_{i_{j-1}}$ has been constructed, the adversary constructs E_{i_j}, by first defining the subsets of P_j, and then constructing $E_{i_{j-1}}$. Let $(S_1, \ldots, S_k)$, $k={}^n/_{c^{j-1}}$ be the subsets of P_{j-1} indexed in increasing order of their potential-degrees in $E_{i_{j-1}}$, then the i^{th} subset of P_j is defined as the union of $S_{(i-1)c+1}, \ldots, S_{ic}$ $i=1,\ldots,{}^n/_{c^j}$. Which implies that if subset S in P_j contains one subset from P_{j-1} whose potential-degree is at least c^j, then all subsets from P_{j-1} in S have potential-degree at least c^j, with the exception of at most one subset of P_j, called the ***boundary*** subset.

In each round in the j^{th} step, $j=1, \ldots, m-1$, we freeze all the subsets in P_j which contain at least one node whose potential-degree $\geq c^j$. When all subsets are frozen step j is finished. The destinations for messages to be sent by node v are selected from the subsets which included v in partitions $P_0, \ldots, P_j$ in that order of priority. This is always possible since every node that has a potential-degree $\geq c^j$ is frozen. Clearly, E_{i_j} satisfies the inductive assumptions. In the m^{th} step no freezing takes place. After that step, the network is contained in one cluster and the algorithm is assumed to produce no more events.

We now show that every node is the destination of at least $^1/_2(c-1)\log_c n$ events in E_{i_m}. Since the time complexity of A is at most $^m/_2$, every node in E_{i_m} must have been frozen in all the rounds of at least $^m/_2$ (complete) steps. Otherwise, the legal-execution which corresponds to E_{i_m} would contain more than $^1/_2\log_c n$ rounds, contradicting the assumption on the time complexity of A. Each node v in a subset which was frozen in all the rounds of step j, is the destination of c-1 messages, one from each of the subsets which joined the subset of v in P_{j-1}, to form P_j. (unless v is in a boundary subset.) The total number of events in E_{i_m} is thus, $\frac{c-1}{\log c} \cdot n \cdot \log n - n \cdot c$. The $n \cdot c$ term is due to the nodes in the boundary subsets. ■

3 A Synchronous Election Algorithm

The algorithm, which is also reported in [5], is initiated by any subset of nodes, each of which is a candidate for leadership. Each candidate tries to capture all other nodes 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 node is elected, all candidates but one are *killed*.

All candidates use a variable called, *age*, to count the number of rounds passed since they have started the algorithm. 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 over 2^i untraversed incident links in an attempt to capture the neighbors at the other end. In rounds $2i+1$ the candidate receives the replies for those messages. Whenever a candidate sees another one with a lexicographically larger $<age, id>$ pair than its own, it kills itself. Clearly, after $2\log n$ rounds there is only one live candidate which has captured all the network and caused the death of any other candidate. The message complexity of the algorithm is $4n \cdot \log n$ (its analysis is omitted from this abstract). Thus proving that the lower bounds are tight.

4 Asynchronous Election Algorithms

Essentially, the same idea of the synchronous algorithm will also work in an asynchronous network. However, unlike the synchronous case, when two candidates of the same *age* meet, the information they have on one another is limited. It might be the case that the candidate with the lower *id* has assumed already a larger *age*. This necessitates the synchronization of candidates by sending messages to each other. The message complexity remains $4n \cdot \log n + O(n)$, but alas, the need for synchronization increases its time to $O(n)$. We postpone the detailed description of the algorithm to the full paper.

In the rest of this section we present a sequence of three algorithms (A, B, and

C) with the aim of reducing the message complexity to $2n \cdot \log n + O(n)$, while at the same time keeping the time complexity linear in n [1]. The underlying mechanism for all three algorithms is similar. Each algorithm is initiated by any subset of nodes, each of which is a candidate for leadership. Each candidate tries to capture all the other nodes by initiating a process which traverses all the links incident to it in both directions. The term *candidate* will be applied interchangeably to both the process and its initiating node. The candidate which has succeeded in capturing all its neighbors becomes the leader. To guarantee that only one node is elected, all candidates but one are *killed.*

Candidate processes use a combination of two variables, *id* and *level*, in order to contest each other. The *id* is the id of the initiator. The *level* is an estimate of the amount of work the candidate has already done. Every captured node has a *level* variable, which is the highest level candidate it has observed. All level variables are initialized to 0.

A candidate that arrives at a node with a larger *level* than its own, kills itself. However, if the candidate's *level* is larger or equal, the node's *level* is replaced by the candidate's *level*. The candidate may then claim the node and try to kill the previous owner of the node.

The main difference between the three algorithms is the way that candidates determine their *level*. In algorithm A, the level of a candidate is the number of nodes it has captured, similar to [7]. In algorithm B, it is the number of candidates it has killed. Algorithm A achieves a better time complexity while algorithm B achieves a better communication complexity. In algorithm C, candidates use a combination of the above two *level* functions to attain the time complexity of A and the message complexity of B.

4.1 Algorithm A

In this algorithm, the *level* of a candidate is the number of nodes it has already captured. When a candidate replaces the *level* of an already captured node, it has to kill the previous candidate which owned the node before it may claim the node to itself. Thus, the sets of captured nodes, each owned by a different live candidate, are disjoint. To prove that the communication complexity of the algorithm is $O(n \cdot \log n)$ we apply a Lemma which was introduced in [7].

Lemma 7: For any given k, the number of candidates that own $^n/_k$ nodes or more is, at most, k.

Lemma 7 gives rise to a $4 \cdot n \cdot \ln n$ ($= 2.773 \cdot n \cdot \log n$) bound on the message complexity. The time complexity of the algorithm is $O(n)$. Its analysis, formal description and correctness were omitted form the abstract.

In algorithm A the number of candidates at a particular *level* is controlled by the disjointness among the sets of nodes that each live candidate owns. This entails the need to "check the situation" with a node's owner, although this owner might not be a candidate any more. Thus, a candidate might have to "kill" the same deceased owner as many times as the number of nodes the latter has captured. In algorithm B that follows we rectify this.

4.2 Algorithm B

In this algorithm whenever a candidate sees another one with a higher level, it kills itself. If a candidate sees another one at the same *level* but with a smaller id, it kills the other candidate and increases its *level* by one. Thus, the *level* of a candidate is the number of candidates it has killed. Unlike algorithm A, a candidate captures a node whose level is smaller than its own without killing the previous owner of the node.

Since, at most half of the candidates at level k proceed to level $k+1$, the maximum *level* achievable during the execution is $\log n$. Clearly, every time a node is recaptured its level is increased by at least one. The total number of messages due to capturings is at most $2n \cdot \log n$, since each capturing incurs two messages. At most $2n$ messages may be incurred by the killings as no more than n killings take place. The time complexity of the algorithm is $O(n \cdot \log n)$. Its analysis and the formal description were omitted from the abstract.

The reason for the high time complexity of algorithm B is that killings are independent of the number of nodes captured by each of the disputing candidates. A candidate which spent a lot of work (and time) capturing nodes might be killed by a candidate which did not capture nearly as much. In the next algorithm we eliminate the problems of both algorithms A and B by making the level a function of both, the number of nodes captured, and candidates killed.

4.3 Algorithm C

Here we make two modifications to algorithm B to achieve a linear time complexity with no increase in the communication cost. First, we combine an estimate of the amount of work spent by each candidate into the level function. Second, we enable candidates with high levels ($>\log n$) to capture many nodes in parallel (in one time unit).

Candidates increase their *level* according to two rules: First, as in algorithm B, a candidate increases its level whenever it kills another one at the same level. Second, candidate sets its *level* to the maximum of *level* and the log of the number of nodes it has already captured.

Lemma 8: The maximum *level* reachable during any execution of algorithm C is $\log n + \log\log n + 1$. (the proof is omitted from the abstract.)

Using the same arguments as in algorithm B we find that the total number of messages spent by algorithm C is bounded by $2 \cdot n \cdot (\log n + \log\log n + 2)$, the length of each message is at most $\log n + \log(\log n + \log\log n)$ bits.

With the above modification it can be shown that the time complexity of algorithm B was reduced to $O(n \cdot \log\log n)$. In order to further reduce the time complexity to $O(n)$, candidates at level higher than $\log n$ will try to capture $^{n}/_{\log n}$ nodes in parallel.

To analyze the modified algorithm we observe that, the maximum attainable level in the algorithm is still bounded by $\log n + \log\log n + 1$, and that a maximum of $\log n$ candidates reach level $\log n$.

The total time delay of the algorithm is bounded by $n + \log n \cdot \log\log n$, its analysis is omitted from the abstract.

5 Conclusions

The effect of synchronous and asynchronous communication on the problem of distributively electing a leader in a complete network was examined. On the one hand, it was proved that the message complexity is not affected by the choice of the communication mode. In both modes, the message complexity was shown to be $\Theta(n \cdot \log n)$. On the other hand, it remains open whether the choice of communication mode affects the time complexity of a message-optimal algorithm. With synchronous communication, the time complexity of message optimal algorithms is proved to be $\Theta(\log n)$, whereas in the asynchronous mode, only an $O(n)$ bound on the time complexity was obtained. The lower bound on time for asynchronous communication remains an open question and is the subject of the following conjecture:
The time complexity of any message optimal asynchronous election algorithm on a complete network is $\Omega(n)$.

The implication of the conjecture is that synchronous communication is faster by a factor of $^n/_{\log n}$ than asynchronous communication. An analogous result was obtained in [2] where a particular synchronous system of parallel processors was proved to be faster by a factor of $\log n$ than the corresponding asynchronous system.

References

[1] 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).

[2] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch, "Efficiency of Synchronous Versus Asynchronous Distributed Systems," *JACM* **30**(3), pp.449-456 (July 1983).

[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] Eli Gafni, Yehuda Afek, and Leonard Kleinrock, "Fast and message optimal synchronous election algorithms for a complete network," CSD-840041, UCLA, Los Angeles, CA 90024 (October 1984).

[6] Eli Gafni, "A note on the problem of electing a leader in a synchronous ring," CSD-85001, University of California, Los Angeles (January 1985).

[7] Robert G. Gallager, "Finding a Leader in a Network with O(E) + O(N log N) Messages," , M.I.T. (1977). Unpublished Note.

[8] 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).

[9] Pierre A. Humblet, "Selecting a Leader in a Clique in O(N log N) Messages," pp. 1139-1140 in *Proceedings of 23rd Conference on Decision and Control*, Las Vegas, Nevada (December 1984).

[10] 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).

[11] 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).

[12] Gary L. Peterson, "Efficient Algorithms for Elections in Meshes and Complete Networks," TR 140, Dep. of Computer Science, Univ. of Rochester, Rochester, New York (August 1984).