Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks †

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 to general algorithms and not just to 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 of which is an improvement on the previous. The third algorithm has time complexity $O(n)$ and message complexity $2 \cdot n \cdot \log n + O(n)$, thus improving the time complexity of the previous best algorithm [Kor84] by a factor of $\log n$.

1. Introduction

In the election problem, a single node, called 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 [Bur80, Fre84, Gal83, Pac82], where n and m are the total number of nodes and links in the network. $\Omega(m)$ is clearly a lower bound, since no algorithm may terminate before sending at least one message over each link, as an untraversed link could be the only link connecting two parts of the network, each holding a separate election. Yet, Korach et. al. [Kor84] 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 asynchronous complete networks, 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 [Fre84] 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 showing that general algorithms are strictly more powerful than comparison algorithms in a ring. This difference stems from the capability of general algorithms in synchronous networks to delay messages and processors as a function of the value of their id's.

In this paper we prove that the message

†This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-82-C-0064, by an IBM Graduate Fellowship, and by an IBM Faculty Development Award.

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 not 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 [Gaf85].

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 a leader in at most $1/2 \cdot \log_c n$ rounds, then its message complexity is at least $\frac{c-1}{2 \cdot \log c} n \cdot \log n$.

Following the lower 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 $5n \cdot \log n$. The asynchronous algorithm is an improvement over the considerably more complicated algorithm in [Kor84] 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 asynchronous 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 [Hum84]) has $O(n)$ time complexity and $2.773 \cdot n \cdot \log n$ message complexity. Algorithm B has $O(n \cdot \log n)$ time complexity but $2 \cdot n \cdot \log n$ 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 $2n \cdot \log n$ message algorithm of [Pet84]. 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 message-optimal election algorithm is $\Omega(n)$.

The models used in this paper are described in Section 2. In Section 3 we present the lower bounds for the synchronous case. Section 4 gives the synchronous algorithm which attains these bounds. Section 5 contains the asynchronous algorithms. Throughout the paper, unless otherwise specified, we use log to denote $\log_2$.

2. The Two Models

In this paper we consider two models: the *synchronous* and the *asynchronous* complete network on n nodes. In such networks every node is connected by $n-1$ bidirectional communication links to all other nodes. Nodes communicate only by exchanging messages. All the links incident to a given node on which *no* message was sent or received are indistinguishable to this node. Each node starts its participation in the algorithm either by being spontaneously awakened at an arbitrary time, in which case it is called an *initiator*, or by receiving a message of the algorithm.

In the synchronous model, a global clock is connected to all nodes. The time interval between two consecutive pulses of the 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. Spontaneously awakened nodes start the algorithm by entering an initial state and then waiting for the beginning of the next round.

In the asynchronous model, there is no global clock and messages incur arbitrary but finite delay. Only when computing the time complexity of an asynchronous algorithm, a constant upper bound on message delay is assumed.

3. Lower Bounds

To show the lower bounds, a scenario in which any synchronous (and hence also asynchronous) algorithm transmits at least $n/2 \cdot \log n$ messages, is constructed using an adversary argument. A similar argument is then used to show that the delay of any message-optimal algorithm is at least $O(\log n)$ rounds.

3.1. Definitions and Assumptions

Consider an arbitrary election algorithm on the synchronous model defined above. 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 separate 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 the algorithm we associate an execution, called a *legal-execution*, that includes all events which occurred in the run, arranged in order of 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 maximum potential-degree among its nodes.

For the purpose of proving the lower bounds we introduce a slightly different model, called the *stopping-model*. The stopping model allows us to withhold the clock pulse, at the beginning of round j from cluster, C, in E_{j-1}, given that no node in C is expected to receive a message in round j from a node not in C. 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. The stopping-model will be used to prevent large differences in the clusters' growth rates.

A *stopping-execution* is an execution which corresponds to a run of the algorithm in the stopping-model. A stopping-execution is called a *k stopping-execution* if the cumulative number of pulses withheld over all clusters throughout the run is k. Obviously, 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 of a round in which any cluster is frozen, and let C be 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 affects neither any event in later rounds nor any event which involves nodes not in C. Because, neither in E nor in E' there is an event connecting a node in C with a node not in C in any round R_j, $j\leq l$, and because R_{l+1} in E is identical to R_{l+1} in E'. Notice that in E' the nodes in C are awakened one round later than in E. ■

Corollary 1: 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. Using Corollary 1, these bounds apply also to the non stopping model. In our proofs we do not restrict the type of operations performed by the nodes, hence proving the bounds for general algorithms.

3.2. A Lower Bound on Message Complexity

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. In other words, by the end of the algorithm the whole network is contained in one cluster. Thus, no cluster in the algorithm can defer indefinitely the sending of messages to nodes not in the cluster, as 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 $1/2n\log n$ 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 1: A stopping-execution of an election algorithm in a synchronous complete network of n nodes contains at least $n/2\cdot\log n$ events, in the worst case.

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

Proof of Theorem 1: Assume w.l.o.g. that $n=2^q$. We define a sequence of partitions $(P_0,\ldots,P_q)$ of the nodes such that each subset in partition P_0 contains one node, and each subset in P_j contains two subsets from P_{j-1}, $1\leq j<q$. Hence, each subset in P_j contains 2^j nodes.

We construct, in q phases, a sequence of stopping-execution-prefixes $(E_{i_0},\ldots,E_{i_q})$, $i_0=0$, 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 node is at least 1. This is done 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, and (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}, $j=1, \ldots, q-1$. In each round of phase j, we freeze all the subsets in P_j whose potential-degree $\geq 2^j$. When all subsets are frozen, phase j is complete. The source and destination nodes of any message sent in this phase 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 q-*th* phase no freezing takes place. After that phase, 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 degree at the end of the algorithm is at least 2^j, for $j=0, \ldots, q-1$. 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 can a message-optimal algorithm be. In the next section we prove that the time complexity of any message-optimal algorithm is $\Omega(\log n)$.

3.3. A Lower Bound on Time Complexity

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

Theorem 2: Any stopping-execution of an election algorithm in a synchronous complete network of n nodes which terminates in less than $1/2 \cdot \log_c n$ rounds, contains at least $\frac{c-1}{2 \cdot \log c} \cdot n \cdot \log n$ events.

Corollary 3: 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 2: Consider an election algorithm whose time complexity is at most $1/2 \cdot \log_c n$. Assume w.l.o.g. that $n=c^q$. A construction similar to the proof of Theorem 1 will be used here. We construct, in q phases, a sequence of stopping-execution-prefixes $(E_{i_0}, \ldots, E_{i_q})$, $i_0=0$, each being a prefix of the next, and a sequence of partitions $(P_0, \ldots, P_q)$, the subset of each partition containing c subsets of the previous. Each subset of P_0 contains one node, thus each subset of P_j contains c^j nodes. E_{i_0} is an empty execution-prefix in which all nodes are awakened spontaneously. 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, 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}. Let $(S_1, \ldots, S_k)$, $k=n/c^{j-1}$ be the subsets of P_{j-1} indexed in nondecreasing order of their potential-degrees in $E_{i_{j-1}}$. Then the i-*th* subset of P_j is defined as the union of $S_{(i-1)c+1}, \ldots, S_{ic}$ $i=1,\ldots,n/c^j$. This 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 of phase j, $j=1, \ldots, q-1$, we freeze all the subsets in P_j whose potential-degree $\geq c^j$. When all subsets are frozen phase j is complete. 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} and P_j satisfy the inductive assumptions. After the q-*th* phase, 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 $1/2 \cdot (c-1) \log_c n$ events in E_{i_q}. As the time complexity is at most $q/2$, every node in E_{i_q} must have been frozen in all the rounds of at least $q/2$ phases. Otherwise, the legal-execution corresponding to E_{i_q} would contain more than $q/2$ rounds, contradicting the assumption on the time complexity. If node v is frozen in all the rounds of phase j, it will later receive one message from every subset in P_{j-1} which do not contain v and is with v in a subset of P_j. (unless v is in a boundary subset). Thus, for each phase that v is frozen in all its rounds, v is the destination of $c-1$ events. The total number of events in E_{i_q} is

thus at least $\frac{c-1}{2\cdot\log c}\cdot n\cdot\log n - n\cdot c$. The term $n\cdot c$ is due to the nodes in the boundary subsets (since due to the boundary subset in phase j at most $c^{j-1}\cdot(c-1)$ events should be discounted). ■

4. The Synchronous Algorithm

In this section we present a $2\cdot\log n$ rounds, $3n\cdot\log n$ messages synchronous algorithm, thus proving that the above lower bounds are tight. An $O(1)$ rounds, $O(n^2)$ messages algorithm is easily constructed by letting every initiator start by sending messages to all its neighbors. All the initiators then elect the initiator with the highest id as the leader. In Section 5 the message complexity of this simple algorithm is reduced to $O(n\cdot\log n)$ by slowing-down the rate in which initiators send messages to their neighbors to one message at a time. However, the reduced rate increases the time complexity of the algorithm to $O(n)$. In this section we use the synchronous model of communication to design an $O(\log n)$ time, message-optimal algorithm. This is done by carefully selecting a variable rate at which initiators send messages to their neighbors.

4.1. Description of the Algorithm

The algorithm 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 succeeds, all candidates but one are *killed*.

To simplify the algorithm every spontaneously awakened node spawns two processes, the *candidate* process and the *ordinary* process. The two processes are connected by a bidirectional logical link which behaves like a physical link. Any other node spawns one ordinary process upon being awakened. Candidate processes communicate only with ordinary processes and vice versa. Thus, the underlying topology is a complete bipartite graph, on one side of which are the candidate processes each of which is connected to the n ordinary processes which reside on the other side. Henceforth, the term *candidate* will be applied interchangeably to both the process and its initiating node.

Every candidate has a variable called *level* which is increased by one every two rounds. Every ordinary process has an *owner-level* and an *owner-id* variable which are the level and id of the highest level candidate it has received a message from (level ties are resolved by selecting the highest id).

In the first round of level i, $i\geq 0$, every live candidate tries to capture 2^i new ordinary processes by sending them messages containing its level and id. If in the second round of level i the candidate receives acknowledgements from all the ordinary processes it tries to capture, it proceeds as a candidate to the next level. On the other hand, if not all the acknowledgements are received, the process (and hence the node owning it) is eliminated form candidacy. In every round, every ordinary process first increases its owner-level by one and then, inspects the newly received messages to update its owner-level and owner-id if necessary. If an update occurred, the ordinary process acknowledges its new owner.

A formal description of the algorithm is given in Figure 1. All messages received by a node are grouped according to their type. Messages sent by candidate processes are forwarded to the ordinary process. Acknowledgements which are generated by ordinary processes are forwarded to the candidate process.

4.2. Time and Message Complexities

Let p be the largest id of a candidate from the set of oldest candidates (i.e., whose level is the largest). We observe the following three facts:

Fact 1: The owner-level 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 network leader.

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

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

Following fact 2, the time complexity of the algorithm is $2\log n$. Since every node sends at most one acknowledgement to a candidate in level i, the total number of acknowledgments is $n\cdot\log n$, each of length $O(1)$ 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 message with

```
/* The candidate process program */

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

/* The ordinary process program */

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

Figure 1: The Synchronous Algorithm

$\log n + \log\log n$ bits.

A continuum of algorithms can be devised to close the gap between the trivial $O(1)$ time, $O(n^2)$ messages algorithm and the $O(\log n)$ time, $3n\cdot\log n$ messages algorithm. Each algorithm in the continuum is the same as the above, except that a candidate in level i 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 $2c\cdot n\cdot\log_c n$, thus proving that the lower bounds of Theorem 2 are tight.

5. Algorithms for Election in Asynchronous Complete Networks

The synchronous algorithm can be easily converted into an asynchronous algorithm. However, because messages incur an arbitrary delay, when two candidates of the same level meet, one could have spent much more time than the other and yet be killed by the other. Therefore it is easy to construct a scenario in which the time complexity is $O(n)$ and the message complexity is $5n\cdot\log n$. Our aim in this section is to derive a $2\cdot n\cdot\log n + O(n)$ messages, linear-time asynchronous algorithm. To this end we present a sequence of three asynchronous algorithms (A, B, and C), each devised to circumvent the problems of the previous, so that algorithm C achieves the desired complexity.

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 spawns a process which tries to capture all the other nodes by successfully traversing in both directions all the links incident to its initiator. 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*.

All candidates use a variable called *level* to estimate the number of nodes they have already captured. The level variable is used by candidates to contest each other. Captured nodes also have a level variable, which tracks the highest level candidate they have observed. All level variables are initialized to 0.

A candidate that arrives at a node with a larger level than its own is eliminated from candidacy. 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 eliminate the previous owner of the node. Upon being killed, the initiating node of a candidate functions like a regular captured node.

The three algorithms differ in two main parts: (1) The way that candidates determine their level, and (2) The rule candidates use to eliminate each other. In algorithm A, the level of a candidate is the number of nodes it has captured (following [Gal77]). In algorithm B, the level is the number of candidates it has killed. Algorithm A achieves a better time complexity while algorithm B achieves a better message 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.

5.1. Algorithm A

Level: In this algorithm, the level of a

candidate is the *number of nodes it has already captured.*

Capturing and Elimination Rule: To capture node v; (1) the (level, id) of a candidate must be lexicographically larger than the (level, id) of the previous owner of v, and (2) the previous owner must be killed.

When candidate P arrives at node v which is currently owned by candidate Q, the following rule is used:

If $(Level(P),id(P)) < (Level(v),id(Q))$, P is killed.
If $(Level(P),id(P)) > (Level(v),id(Q))$, (1) v gets P's level, and (2) P is sent to Q.

Upon arriving to Q:
If $(Level(P),id(P)) < (Level(Q),id(Q))$, P is killed.
If $(Level(P),id(P)) > (Level(Q),id(Q))$, then (1) Q is killed or Q has been killed already, and (2) P captures v.

Upon returning to its initiating node from a successful capturing, P increases its level by one.

Details

To keep track of its owning candidate, every captured node has two link pointers, *father* and *potential–father*. The father pointer points to the link through which the node was most recently captured, and the potential-father pointer points to the link through which a candidate which tries to claim the node from its father, has arrived.

A candidate that arrives at an already captured node v whose level is smaller than its own, replaces v's level with its own and becomes v's *potential–father*. The potential-father candidate is then sent to the father candidate of v. If the candidate survives at v's father, and no other candidate becames $v's$ potential-father in the meanwhile, then it becomes $v's$ father. If v has not yet been captured, the potential-father automatically becomes the father of v.

Analysis

The algorithm is deadlock-free since candidates never wait for each other, and the (*level*, *id*) pair is lexicographically increasing along any chain of candidates which kill each other.

The time complexity of the algorithm is $O(n)$ since candidates never wait for each other and a candidate which has done more work is never killed by a candidate which has done less work. Thus, each killed candidate spent, in the worst case, less time than the one killing it.

To prove that the communication complexity of the algorithm is $O(n \cdot \log n)$ we use a Lemma which was introduced in [Gal77].

Lemma 2: For any given k, the number of candidates that own n/k or more nodes is at most k.

Proof: Let C_1 and C_2 be any two candidates which owned n/k nodes at some point of time. We shall show that each of C_1 and C_2 must have owned at least n/k nodes disjointly. If they never tried to claim a node from each other, we are done. The first time that C_1 (w.l.o.g.) tries to claim a node, say v, from C_2, either it causes the death of one of them, or C_2 has been already killed. If C_1, w.l.o.g., caused the death of C_2 then clearly it must have owned at least n/k nodes disjoint from C_2, at the time of killing. If C_2 is already dead, C_1 must still own at least n/k nodes in order to claim v to itself. ■

Corollary 4: The largest candidate to be killed by another candidate owns at most $n/2$ nodes, the next largest owns at most $n/3$ nodes, etc.

Lemma 3: The message complexity of algorithm A is $4 \cdot n \cdot \ln n$ $(2.773 \cdot n \cdot \log n)$ messages.

Proof: Since in capturing one node a candidate makes at most 4 hops, a candidate which owned k nodes incurs at most $4 \cdot k$ messages. By Corollary 4, the total cost is then bounded by $4 \cdot n \cdot \sum_{i=1}^{n} \frac{1}{i}$ messages. Note that each message of the algorithm contains at most $2 \cdot \log n$ bits. ■

The number of candidates at a particular level was constrained by the disjointness property. Hence, a candidate which captures many nodes from another candidate, tries to eliminate that other candidate as many times as the number of nodes it captures from it. This gives rise to the factor 4 in the message complexity. In the next algorithm we remove the disjointness requirement and change the level function to reduce the message complexity to $2 \cdot n \cdot \log n$ messages.

5.2. Algorithm B

Level: In this algorithm, the level of a candidate is the *total number of other candidates that it has killed.*

Capturing and Elimination Rule: To capture node v the level of a candidate must be strictly larger than that of v, in which case the candidate captures v without killing the previous owner of v.

When candidate P arrives at node v which is currently owned by candidate Q, the following rule is used:

If $Level(P) < Level(v)$, P is killed.
If $Level(P) > Level(v)$, v is captured by P, and v gets P's level.

If $Level(P) = Level(v)$, P is sent to Q.
Upon arriving to Q:
If $(Level(P),id(P)) < (Level(Q),id(Q))$, P is killed.
If Q has already been killed, P is killed too.
If $(Level(P),id(P)) > (Level(Q),id(Q))$, then (1) Q is killed, (2) P increases its level by one, and (3) P captures v.

Details

When a candidate arrives at node v whose level is the same as its own, and the id of v's father, Q, is smaller, it becomes v's potential-father. The potential-father is then sent to Q in an attempt to kill it. If another candidate at the same level with even higher id arrives at v before the potential-father returns from Q, then this other candidate is killed. If the potential-father survives at Q it first increments its level by one, then returns to v and captures it, and only then, returns to its initiating node. However, if the potential-father finds that Q is already killed, it eliminates itself as well (since if Q was killed, there exists a higher level candidate in the network).

Analysis

Since at most half of the candidates at level k go up to level $k+1$, the maximum level achievable during the algorithm is $\log n$. Clearly, every time a node is recaptured its level is increased by at least one. Hence, the total number of capture messages possible is at most $n \cdot \log n$. Each capture uses 2 messages, which sums up to a total of $2 \cdot n \cdot \log n$ messages. The extra messages spent by candidates which go over father links to other candidates is at most $2 \cdot n$, since each such traversal results in the elimination of one live candidate. Thus, the message complexity of the algorithm is $2 \cdot n \cdot \log n + 2 \cdot n$ messages, each of length $\log n + \log\log n$ bits.

The time complexity of the algorithm is $O(n \cdot \log n)$ by the following scenario, in which $n/2$ of the nodes are captured serially $\log(n/2)$ times. The algorithm is started by node v_0 which captures $n/2$ nodes in level 0. Then, a new node, v_1, spontaneously starts the algorithm, kills v_0, increases its level to 1 and recaptures the same $n/2$ nodes. After v_1 has captured the $n/2$ nodes, two new nodes spontaneously start the algorithm, try to kill each other, and the one which survives, v_2, reaches level 1. Node v_2 then kills v_1 and recaptures the $n/2$ nodes at level 2. The scenario continues until the entire network has been captured by $v_{\log n/2}$ which is elected as a leader.

The increase in time complexity of the algorithm is because unlike algorithm A, the level of a candidate here is not a function of the number of nodes it has already captured. A candidate which spent a lot of work (and time) accumulating nodes might be killed by a candidate which did not spend nearly as much. Although algorithm A does not suffer from this problem, it has the problem that candidates could be "killed" many times. In the next algorithm we eliminate both problems by employing both techniques simultaneously in one algorithm.

5.3. Algorithm C

Here we make two modifications to algorithm B in order to achieve a linear-time complexity with no increase in the communication cost. First, we incorporate an estimate of the amount of work spent by each candidate into the level function of algorithm B. Second, we enable candidates with a high level ($>\log n$) to capture many nodes in parallel (in one time unit). We start describing the algorithm with the first modification. The second modification will be introduced during the performance analysis.

Level: In this algorithm the level of a candidate is increased according to two rules. First, the same rule as in algorithm B is used, and second after each capturing the candidate increases its level to be at least log(*total number of nodes captured*), i.e., after returning from a successful capture the level is set to *MAX(log (# nodes captured), present level).*

Capturing and Elimination Rule: Same as in algorithm B.

Analysis

To analyze its performances we will first show that:

Lemma 4: The maximum *level* reachable during any execution of algorithm C is $\log n + \log\log n + 1$.

Proof: Let N_i be the total number of candidates that reach level i during the execution of the algorithm. Consider the maximum number of candidates which could possibly pass from level i–1 to level i. There are two ways in which a candidate can go from level i–1 to level i. First, by capturing 2^{i-1} nodes at level i–1 for $i \leq \log n$, and second, by killing another candidate which is at level i–1. We note that N_i is maximized if as many candidates as possible pass from level i–1 to level i by capturing other nodes (i.e., $\frac{n}{2^{i-1}}$ candidates) and the rest of the candidates (i.e, $N_{i-1} - \frac{n}{2^{i-1}}$) kill each other in pairs. Hence,

$$N_i \leq \frac{(N_{i-1} - \frac{n}{2^{i-1}})}{2} + \frac{n}{2^{i-1}} \qquad (1)$$

Solving (1) for N_i we get:

$$N_i \leq \frac{n}{2^i} \cdot (i+1) \qquad (2)$$

Substituting $N_i = 1$ in (2) and solving for i gives us the maximum level, which is $\log n + \log\log n + 1$. ■

Using the same argument as in algorithm B we find that the message complexity of algorithm C is $2 \cdot n \cdot (\log n + \log\log n + 2)$ messages, each of length $\log n + \log(\log n + \log\log n)$ bits.

With the above modification it can be shown that the time complexity of algorithm B is reduced to $O(n \cdot \log\log n)$. In order to further reduce the time complexity to $O(n)$, processes at level higher than $\log n$ will try to capture $n/\log n$ nodes in parallel. Thus a candidate which has reached level $\log n$ will send messages over $n/\log n$ untraversed links incident to it. Each of these messages carry the (level, id) of the candidate. When a message arrives at an adjacent node the node compares its level to that of the message. If the message level is higher, the node replaces its (level, id) with the message, thus making the candidate the new father of the node. The node then sends the candidate an acknowledgement of successful capture. If the message level is smaller, it returns no message. Finally, if the message level is the same as that of the node but the message id is higher, a notification to that effect is sent back to the candidate.

The candidate waits for all the $n/\log n$ acknowledgements. If all the acknowledgements indicate a successful capture, the candidate proceeds to the next $n/\log n$ untraversed incident links. If, on the other hand, some of the acknowledgements indicate that they have encountered the same level, one of the links is arbitrarily chosen and a process that behaves as in algorithm B is sent along that link. If the process returns, the candidate increases its level and proceeds to the next $n/\log n$ untraversed links (links on which no successful capture was reported are not considered traversed).

To analyze the algorithm with this modification we make two observations: First, the maximum attainable level in the algorithm is still bounded by $\log n + \log\log n + 1$. Second, by substituting $i = \log n$ in equation (2), we find out that the maximum number of candidates which reach level $\log n$ is $\log n$.

The last modification has increased the communication complexity of the algorithm by at most $O(n)$ messages. Each node is still captured at most $\log n + \log\log n + 1$ times, however the death of a candidate at level greater than $\log n$ might be associated with at most $2 \cdot n/\log n$ messages. Since there are at most $\log n$ such candidates the increase due to killings is bounded by $O(n)$.

To show that the time complexity of the algorithm is $O(n)$ we arrange the candidates in a rooted tree. Each level of the tree corresponds to the candidates that have reached that level in the algorithm, i.e., the nodes at level i in the tree correspond to the candidates that have reached level i in the algorithm. The parent of a candidate at level i in the tree is either the candidate that caused the death of the given candidate or, the same candidate at the next level. The time delay of the algorithm is the sum of the delays incurred by candidates along the path from the first candidate spontaneously waking up, at level 0, to the root.

To evaluate this time delay we note that no candidate that either survives or is killed at level i spends more than 2^{i-1} time units in level i, $i \leq \log n$. In level i, $\log n < i \leq \log n + \log\log n$ no candidate spends more than $\log n$ time units since it captures nodes at a rate of $n/\log n$ per time unit. Hence, the total time delay of the algorithm is bounded by $\sum_{i=1}^{\log n} 2^i + \log n \cdot \log\log n = n + \log n \cdot \log\log n$. Note that we scale a time unit to be the maximum delay it takes to capture one node, which is a constant.

In the above calculation we did not include the actual time it takes candidates to kill each other. Since there are at most n candidates and no candidate tries to kill a dead one (unlike algorithm A), this delay is also bounded by $O(n)$.

6. 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 of communication, the message complexity was shown to be $\Theta(n \cdot \log n)$. On the other hand, it remains open whether or not the choice of communication mode affects the time complexity of a message-optimal algorithm. With synchronous communication, the time complexity of message-optimal algorithms was proved to be $\Theta(\log n)$, whereas with asynchronous communication, only an $O(n)$ upper 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:

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 [Arj83], where a particular synchronous system of parallel processors was proved to be faster by a factor of $\log n$ than the corresponding asynchronous system.

In section 5, three asynchronous election algorithms (A, B, and C) were presented. The simplicity of the complete network topology, and, hence, of termination detection, enabled us to concentrate on the synchronization among contending candidates. With each of the three algorithms we can associate an analogous algorithm for arbitrary topology networks, which uses the corresponding method to synchronize different initiations of the algorithm but a different method to traverse the network (i.e., to detect termination). The analogous of algorithm A is given in [Gal77]. In [Gal83] the same level function as in algorithm B was used, however, there candidates merge their "territories" (rather than kill each other) when they meet. In [Gaf85] the time complexity of [Gal83] is improved by replacing its level function with that of algorithm C. Each of the methods can be applied to other classes of networks. For example, applications of method B in different classes of topologies are presented in [Kor85]. As in algorithm B, the time complexity of the algorithms in [Kor85] is $O(n \cdot \log n)$, whereas similar applications, but of methods A or C, improve the time complexity of [Kor85] while maintaining the message optimality.

References

[Arj83] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch, "Efficiency of Synchronous Versus Asynchronous Distributed Systems," *JACM* **30**(3), pp.449-456 (July 1983).

[Bur80] J. E. Burns, "A Formal Model for Message Passing Systems," TR-91, Indiana Univ., Bloomington (May 1980).

[Fre84] 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).

[Gaf85] Eli Gafni, "Improvements in the Time Complexity of Two Message-Optimal Election Algorithms," in *Proceedings of the ACM Symp. on Principles of Distributed Computing*, Minacki Ontario (August 1985). also UCLA CSD-85001 January 1985.

[Gal77] Robert G. Gallager, "Finding a Leader in a Network with O(E) + O(N log N) Messages," , M.I.T. (1977). Unpublished Note.

[Gal83] 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).

[Hum84] 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).

[Kor84] 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).

[Kor85] E. Korach, S. Kutten, and S. Moran, "A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms," in *Proceedings of the ACM Symp. on Principles of Distributed Computing*, Minacki Ontario (August 1985).

[Pac82] 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).

[Pet84] 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).