Leader Election in Complete Networks

Gurdip Singh
Department of Computing and Information Sciences,
Kansas State University, Manhattan, KS 66506
singh@cis.ksu.edu

Abstract: This paper presents protocols for leader election in complete networks. The protocols are message optimal and their time complexities are a significant improvement over currently known protocols for this problem. For asynchronous complete networks with sense of direction, we propose a protocol which requires $O(N)$ messages and $O(logN)$ time. For asynchronous complete network without sense of direction, we show that $\Omega(N/logN)$ is a lower bound on the time complexity of any message optimal election protocol and we present a family of protocols which requires $O(Nk)$ messages and $O(N/k)$ time, $logN \leq k \leq N$. Our results also improve the time complexity of several other related problems such as spanning tree construction, computing a global function, etc.

1 Introduction

In the leader election problem, there are N processors in the network, each having a unique identity. Initially all nodes are *passive*. An arbitrary subset of nodes, called the *base* nodes, wake up spontaneously and start the protocol. On the termination of the protocol, exactly one node announces itself the leader. In this paper, we consider the problem of electing a leader in an asynchronous complete network. In a complete network, each pair of nodes is connected by a bidirectional link and we assume that a node is initially unaware of the identity of any other node.

Leader election is a fundamental problem in distributed computing and has been studied in various computation models. For complete networks in which a node is unable to distinguish between its incident links, [KMZ84] showed that $\Omega(NlogN)$ messages are required for electing a leader. However, [LMW86] showed that the lower bound of $\Omega(NlogN)$ messages does not hold for complete networks with sense of direction and gave a protocol which requires $O(N)$ messages. A network has a *sense of direction* if there exists a directed Hamiltonion cycle and each edge incident at any node i is labeled with the distance of the node at the other end along this Hamiltonion cycle. Figure 1 shows a complete network containing six nodes with a sense of direction. [ALSZ89] further showed that $O(logN)$ chords in a ring network are sufficient to obtain a protocol with $O(N)$ message complexity. These two extreme cases, one in which a node is unable to distinguish between any two incident edges and the other in which all edges are labeled with a distinct number, show the impact of knowledge of topological information on the complexity of leader election.

The time complexity of the protocol in [LMW86] is $O(N)$. In this protocol, a node captures a majority of nodes before it declares itself as the leader. We observe that in such networks, a node does not have to capture a majority of nodes in order to be elected the leader. We use this idea to obtain a simple protocol which requires $O(N)$ messages. However, due to congestion on the links and a specific wake up pattern of nodes, its time complexity is not $O(logN)$. We then modify this protocol to solve these problems and obtain a protocol which requires $O(N)$ messages and $O(logN)$ time.

We also propose an improved protocol for leader election in asynchronous complete network without sense of direction (in the rest of the paper, unless other stated, we will use 'complete network' to mean 'complete network without sense of direction'). [KMZ84] proposed a protocol for this problem which requires $O(NlogN)$ messages and $O(NlogN)$ time. [AG85] gave a series of simple message optimal protocols for complete networks, each with $O(N)$ time complexity. Furthermore, it was conjectured in [AG85] that $\Omega(N)$ is a lower bound on the time complexity of any message optimal election protocol for asynchronous complete networks. We prove that $\Omega(N/logN)$ is a lower bound on the time complexity of any message optimal protocol for this problem. This proves that introducing asynchrony may result in a loss in speed by a factor of $N/(logN)^2$. A similar result was shown in [AFL83] where a particular asynchronous system was shown to be slower by a factor of $logN$ than the corresponding synchronous system. We also provide a message optimal protocol for asynchronous complete networks which requires $O(N/logN)$ time. The protocol involves a new technique which allows us to distinguish between nodes that wake up at different times to participate in the protocol. The complexity of the protocol depends on the number of base nodes and we show that the time complexity can be improved to $O(logN + min(r, N/logN))$, where r is the number of base nodes. We also present a protocol tolerant to f initial site failures which requires $O(Nf + NlogN)$ messages and $O(N/logN)$ time, where $f < N/2$.

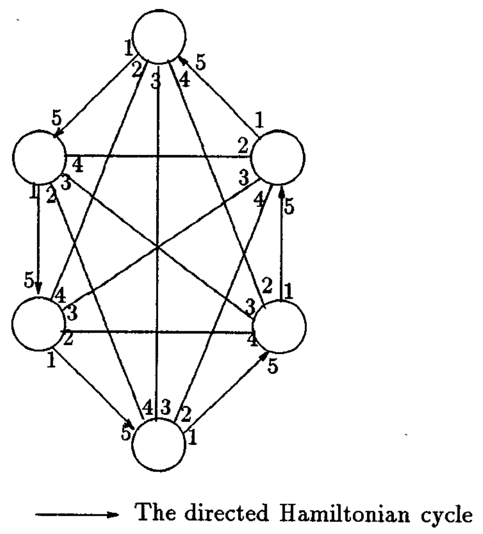

Figure 1: A complete network with a sense of direction

There are many problems such as spanning tree construction, computing a global function, etc. which are equivalent to leader election in terms of message and time complexities. Our protocols, therefore, leads to improvement in the time complexity of these problems as well.

This paper is organized as follows. In the next section, we present our model of distributed computation. In Section 3, we present a protocol for leader election in a complete network with sense of direction. In Section 4, we present a protocol for leader election in a complete network without sense of direction and we show a lower bound on the time complexity of any message optimal leader election protocol.

2 Model

We model the communication network as a complete graph (N, E), where N and E represent the processors and communication links respectively. We assume that each node has a unique identity. Messages sent over a link arrive at their destination within finite but unpredictable time and in the order sent and are not lost. Each message may carry $O(logN)$ bits of information. The message complexity of a protocol is the maximum number of messages sent during any possible execution of the protocol. The time complexity of a protocol is the worst case execution time assuming that each message takes at most one time unit to reach its destination and computation time is negligible. Furthermore, inter-message delay on a link is at most one time unit. All additions in the paper are assumed modulo N.

3 Complete Networks with sense of direction

For complete networks with sense of direction, [LMW86] proposed a leader election protocol which requires $O(N)$ messages and $O(N)$ time. Let $i[d]$ denote the node at distance d from i and $i[x..y]$ denote the set $\{i[x], i[x+1], \ldots, i[y]\}$. In [LMW86], if node i is able to capture nodes in $i[1..N/2]$ then it can declare itself as the leader. We observe that in the presence of a sense of direction, a node does not have to capture a majority of nodes. For example, if i captures all nodes in $\{i[1..N/4], i[N/2], i[3N/4]\}$ then it can declare itself as the leader. By capturing $i[N/2]$, for example, i ensures that no node in $i[N/4+1..N/2]$ will be able to become a leader since a node in this set must capture $i[N/2]$ to become a leader. In particular, a node can declare itself as the leader after capturing the nodes in the set $\{i[1..k], i[2k], i[3k], \ldots, i[N-k]\}$. We combine this idea with those in [LMW86] to obtain a new protocol, $\mathcal{A}$, which is as follows:

Protocol $\mathcal{A}$: This protocol proceeds in two phases:

First Phase: On waking up spontaneously, a base node i tries to capture nodes in $S_i = i[1..k]$ in a sequential fashion. A *passive* node wakes up on receiving a message of the protocol. A *passive* node is not allowed to become a base node if it wakes up on receiving the message of the protocol. A base node i uses its identity and $level_i$, which denotes the number of nodes which i has captured so far, to contest with other nodes. When a base node i wakes up, it sends a message $capture(i, level_i)$ to $i[1]$. When a node j receives a $capture(i, l)$ messages, it behaves as follows:

- If j is not a base node or it has been captured then it responds with an $accept(0)$ message.
- If j is a base node which has not yet been captured, and $(level_j, j) < (l, i)$ then again, i captures j and j responds with $accept(level_j)$. Otherwise, j ignores the message.

If i receives $accept(l)$, it adds $l+1$ to $level_i$ (and therefore the set of captured nodes is extended to include the nodes captured by j). If $level_i < k$ then it continues its conquest by sending a *capture* message to $i[level_i + 1]$. Otherwise, it enters the second phase.

Second Phase: On entering this phase, i sets $owner_i$ to i and sends a message, $owner(i)$, to each node j in $i[1..k]$. On receiving this message, j sets $owner\text{-}link_j$ to denote the link from j to i and $owner_j$ to i. Furthermore, it sends an acknowledgement message to i. After receiving an acknowledgement from all nodes in $i[1..k]$, i sends an $elect(i)$ message to each node in $\{i[2k], \ldots, i[N-k]\}$. On receiving $elect(i)$, site

j behaves as follows: If ($owner_j$ has not been set) or ($owner_j$ has been set and $owner_j < i$) then it sets $owner_j$ to i and sends an *accept* message to i. Otherwise, it ignores the message. If i receives all accept responses then it declares itself the leader.

In the first phase, each node is captured at most once and each capturing requires one *capture* message and one *accept* message. Hence, the first phase will require $O(N)$ messages. In the second phase, there can be at most $O(N/k)$ candidates. Each candidate sends messages to capture N/k nodes. Hence, the total number of messages in the second phase is $O(N^2/k^2)$. The message complexity of $\mathcal{A}$ is therefore $O(N + N^2/k^2)$. In particular, for $k \geq \sqrt{N}$, the protocol requires $O(N)$ messages. We will now compute the time complexity of $\mathcal{A}$. The execution of the second phase takes $O(1)$ time. Furthermore, if a node is successful in capturing another node then it does in a constant amount of time. Hence, the node which is elected the leader will finish its first phase within $O(k)$ time units of waking up. However, the following situation can arise: Assume that nodes have identities $1, \ldots, N$ such that $i[1] = i + 1$. Let 1 be the first node to wake up. After waking up spontaneously, node i sends a *capture* message to node $i + 1$. $i + 1$ wakes up just before the message from i reaches it and sends the message to capture $i + 2$ before receiving the message from i. In this case, no response will be sent to i since i has the same level number as i+1 but a smaller identity. If this happens for all sites i, $1 \leq i \leq N$, then only node N will survive and capture all other nodes. If the capture message for each node takes exactly one time unit to arrive, node N will wake up at time $N - 1$ and therefore the protocol will require $O(N)$ time units. However, *if all nodes wake up within $O(k)$ time of each other then the first phase will take $O(k)$ time.* We will now modify the protocol $\mathcal{A}$ to obtain $\mathcal{A}'$ as follows: After a node i wakes up (either spontaneously or on receiving a message), it sends a message to awaken $i[1]$ and $i[k]$. Hence, within $O(k + N/k)$ time, all nodes will wake up to participate in the protocol. Therefore, the time complexity of the protocol is $O(k + N/k)$. In particular, for $k = \sqrt{N}$, the time complexity of $\mathcal{A}'$ is $O(\sqrt{N})$.

We will now extend this idea to obtain a protocol which requires $O(logN)$ time. Consider the following protocol, $\mathcal{B}$, which is an asynchronous version of the synchronous protocol in [AG85]. For simplicity, assume that N is of the form 2^x. In this protocol, a candidate node i tries to capture all other nodes in $logN$ steps. In the first step, i sends a message to capture $i[N/2]$. In the l^{th} step, i sends a message to capture $2^l - 1$ nodes in the set $i[N/2^l], i[3N/2^l], \ldots, i[(2^l - 1)N/2^l]$. If $i[N/2]$ is also a base node then it will send a message to capture i in its first step. Hence, only one of i and $i[N/2]$ will proceed to step 2. Similarly, only one of i, $i[N/4]$, $i[N/2]$ and $i[3N/4]$ will proceed to step 3 and so on. Although the time complexity of this protocol is $O(logN)$, its message complexity is $O(NlogN)$.

We will now combine ideas in $\mathcal{A}$ and $\mathcal{B}$ to obtain a protocol $\mathcal{C}$ which has $O(N)$ message complexity and $O(logN)$ time complexity. $\mathcal{C}$ proceeds in two phases. In the first phase, we use $\mathcal{A}$ to first reduce the number of candidates to at most $N/logN$. The second phase employs $\mathcal{B}$ to elect the leader. Let $k = N/2^{\lceil loglogN \rceil}$.

First Phase: In this phase, i tries to capture $i[k], i[2k], \ldots, i[N - k]$ in a sequential manner. Observe that when $i[xk]$ wakes up, it will try to capture the same set of nodes in the order $i[xk + k], \ldots, i[xk + N - k]$. Hence, in the first phase, nodes in this set compete against each other. The rules for capturing are the same as in the first phase of $\mathcal{A}$. Hence, for example, if $i[xk]$ has already captured $i[(x + 1)k]$ when i captures $i[xk]$, then $i[xk]$ surrenders $i[(x + 1)k]$

to i and therefore, i can extend its set of captured nodes to include this node also. Using i as the reference node, the nodes can be partitioned into k sets, $R_0, \ldots, R_{k-1}$, where $R_j = \{i[j+k], i[j+2k], \ldots, i[j+N-k]\}$. After the first phase, we have at most one alive candidate from each of these sets.

Second Phase: On entering this phase, node i sends messages to each node j in R_i to update $owner_j$ to i. In this phase, we have at most one candidate in each set R_j, and we have to elect a leader among them. Let i be the candidate in R_0. In order to defeat other candidates, i tries to capture the nodes in the set $i[1..k-1]$ in $logk$ steps. This phase is an asynchronous version of the synchronous protocol in [AG85]. In the first step, i sends an $elect(i, 0)$ message to capture $i[k/2]$. In the l^{th} step, it sends 2^{l-1} messages to capture $i[k/2^l], i[3k/2^l], \ldots, i[(2^l-1)k/2^l]$. Observe that if there is an alive candidate in $R_{k/2}$ then it will send a message to capture a node in R_0 in its first step. Hence, only one node from R_0 and $R_{k/2}$ will go to step 2. In step 2, since a node sends messages to capture nodes at distance $k/4$ and $3k/4$, only one node from R_0, $R_{k/4}$, $R_{k/2}$ and $R_{3k/4}$ will survive. In general, after the l^{th} step, there will be at most $k/2^l$ alive candidates (note that k is $O(N/logN)$). In this phase, to contest with other nodes, i uses its identity and $step_i$, which indicates the number of steps which i has executed so far. Consider the case in which a node, i, in R_x sends a message to capture a node, j, in R_y. If j is passive then there is no base node in R_y which has captured all nodes in R_y and therefore, j sends an *accept* message to i. If there is a candidate in R_y then the message is forwarded to that node ($owner\text{-}link_j$ will be the edge leading to this node) and they compete on the basis of $(step, id)$. However, if $i's$ message reaches the candidate in R_y and finds that this node has already been captured then i must first kill the owner of R_y's candidate before it can claim R_y's candidate. For this purpose, the message is forwarded to that node (thus, each message can be forwarded at most twice). For example, if i in $R_{k/2}$ sends a message to capture j in $R_{3k/4}$, and the base node in $R_{3k/4}$ has already been captured by $R_{k/4}$ in step 1 then i must defeat the base node in $R_{k/4}$ before claiming $R_{3k/4}$.

The first phase requires $O(logN)$ time since a node competes only with $O(logN)$ other nodes. The second phase involves $O(logN)$ steps, each of which will take a constant amount of time. Hence, the protocol requires $O(logN)$ time. We will now compute the message complexity of $\mathcal{C}$. The first phase requires $O(N)$ messages since a node is captured at most once. In the second phase, there can be at most k candidates. Furthermore, *there can be at most $k/2^{l-1}$ nodes in step l* (since a node in step l must have captured 2^{l-1} nodes and sets of nodes captured by different sites are disjoint). A node in step l sends 2^{l-1} messages to capture nodes. Each of these messages generate a constant number of messages. Since $k = O(N/logN)$, the total number of messages generated in the second phase is $\sum_{1 \le l \le logN}(k/2^{l-1} * O(2^{l-1})) = \sum_{1 \le l \le logN}(O(N/(logN * 2^{l-1})) * O(2^{l-1})) \le O(logN * N/logN) = O(N)$. Hence, the message complexity of the protocol is $O(N)$.

4 Complete Networks without sense of direction

In this section, we will present a family of algorithms for leader election in complete networks without sense of direction. Protocols belonging to this family require $O(Nk)$ messages and $O(N/k)$ time, where $logN \le k \le N$ [Si91]. We will first present two different algorithms, $\mathcal{D}$ and

$\mathcal{E}$, for leader election. We will then combine features of these algorithms to obtain the final protocol $\mathcal{F}$.

Protocol $\mathcal{D}$: In this algorithm, a *base* node attempts to capture all other nodes in parallel. On waking up spontaneously, a base node sends its identity in an *elect* message on all incident edges. When a node j receives an *elect*(i) message over edge e, it behaves as follows: If j is a base node and $j > i$ then no response is sent over e; Otherwise, j sends an *accept* message over e. A node that receives an *accept* message on all incident edges declares itself the leader. The time complexity of this protocol is O(1). However, its message complexity is $O(N^2)$ since the number of base nodes may be $O(N)$, each of which will send $O(N)$ messages.

Protocol $\mathcal{E}$: $\mathcal{E}$ is a modification of the protocol **A** in [AG85]. The outline of protocol **A** in [AG85] is as follows:

> A base node tries to capture other nodes in a sequential manner by sending *capture* messages on its incident edges one at a time. A node that is successful in capturing all other nodes is elected the leader. A base node i sends its identity and a variable, $level_i$, in the *capture* message to contest with other nodes ($level_i$ is the number of nodes which i has captured so far). If a capture message from i reaches a node j which has not yet been captured, and $(level_i, i) \geq (level_j, j)$ (lexicographically) then i captures j, otherwise i is killed. If j is a captured node, then i has to kill j's owner before claiming j. If i is successful in capturing j then it increments $level_i$ and proceed with its conquest by sending a capture message to another node.

The message complexity of **A** is $O(NlogN)$. Although the time complexity of **A** is $O(N)$, it does not possess the property that a node is able to capture another node in a constant amount of time. For example, a captured node j may receive *capture* messages from nodes $i_1, i_2, \ldots, i_m$ in the order given and forward each of these messages to its owner. In particular, if it forwards $O(N)$ messages and only the last forwarded message is able to defeat $owner_j$ then it may take $O(N)$ time to capture j (since the messages are forwarded on the same link and inter-message delay on the same link can be 1 time unit, the last forwarded message may reach j after $O(N)$ time units). We modify **A** to obtain $\mathcal{E}$ in which there is at most one forwarded message on a link at any time. In $\mathcal{E}$, a captured node j uses a boolean variable $forward_j$ to keep track of whether or not it has forwarded a message to its owner. If j receives a capture message and $forward_j$ is true then it delays forwarding the message to its owner until it receives a response from its owner. Each message forwarded to the owner is responded by an accept or a reject message depending on whether the forwarded message defeated the owner. If an accept message is received then j sends an accept message to the node from which it has received the largest $(level, id)$ pair so far. If a reject message is received, j forwards the message with the largest $(level, id)$ pair it may have received in the meanwhile. Thus, in $\mathcal{E}$, if a node is able to capture another node then it does so in a constant amount of time.

In $\mathcal{D}$, if the number of candidates is restricted to $O(k)$ then it will require $O(Nk)$ messages. In protocol $\mathcal{E}$, there can be at most k nodes at level N/k [AG85]. We obtain a new protocol $\mathcal{F}$ in which $\mathcal{E}$ is used to reduce the number of candidates for protocol $\mathcal{D}$ by requiring a node to execute $\mathcal{E}$ until its level number reaches N/k and $\mathcal{D}$ thereafter, where $logN \leq k \leq N$. The protocol $\mathcal{F}$ is as follows:

> On waking up spontaneously, a node starts executing protocol $\mathcal{E}$. When a node reaches level N/k, it sends an *elect* message with its identity on all incident edges. Let node

j receive an *elect*(i) message over e. If $(level_j, maxid_j)$ is less than $(N/k, i)$ then j changes status to *killed* and sends an *accept* message over e. A node which receives an *accept* message on all incident edges declares itself the leader.

Since the message complexity of $\mathcal{E}$ is $O(NlogN)$ and at most k nodes broadcast an *elect* message, the message complexity of $\mathcal{F}$ is $O(Nk)$ (since $k \geq logN$). Since it takes a constant amount of time to capture a node, it will take $O(N/k)$ time for a node to reach level N/k after it wakes up (if it reaches this level). After a node reaches this level, it executes $\mathcal{D}$ which takes $O(1)$ time. Therefore, the node which is elected the leader will take $O(N/k)$ time after waking up spontaneously to declare itself the leader. Thus, we have the following lemma:

Lemma 4.1 *If all nodes wake up within $O(N/k)$ time of each other, then $\mathcal{F}$ will terminate in $O(N/k)$ time.*

However, a situation similar to the one in the first-phase of protocol $\mathcal{A}$ for networks with sense of direction (in which $i+1$ wakes up just before the message from i reaches it) can occur which may lead to an execution in which the node which is elected the leader wakes up $O(N)$ time units after the first node wakes up. Since nodes are unable to distinguish between the incident edges, the solution used in the presence of sense of direction will not work here. However, we also have the following result: After a node i reaches level k, only a node at level at least k can capture it. Hence, in every interval of c time units after a node reaches level k, where c is a constant, either the node with the highest level number (which is greater than k) will increase its level number or it will be killed by another node with level number at least k. Since there are at most N/k nodes at level k, some node will reach level N/k within $2cN/k$ time. Thus, we have the following lemma:

Lemma 4.2 *After a node reaches level k, $\mathcal{F}$ terminates within $O(N/k)$ time.*

In the following, we will design a protocol, $\mathcal{G}$, in which we will ensure that *in every interval of c time units, either at least k nodes wake up or some node reaches level at least k.* Then from Lemma 4.1 and Lemma 4.2, the protocol will require $O(N/k)$ time. For this purpose, we require a base node to execute two initial phases on waking up spontaneously. If it successfully executes these phases, it qualifies as a candidate for election and proceeds by executing $\mathcal{F}$. Intuitively, the time complexity of the leader election protocol depends on the ability to recognize the order in which nodes wake up to participate in the protocol so that nodes that wake up later are prohibited from becoming candidates. In the first phase, a node tries to obtain permission from k other nodes. If i requests permission from j after j has finished executing its first phase, it denies permission to i. In this case, i gets ordered after j and is not allowed to participate as a candidate. However, as we will show later, this allows a node which wakes up $O(N/k)$ time units after the first node wakes up to obtain permission from k other nodes and participate as a candidate. The first-phase is as follows:
On waking up spontaneously, node i selects k incident edges and sends a *first-phase*(i) message on each of these edges. On receiving this message, site j behaves as follows:

- If j is not a captured node then
  if j has finished executing its first phase then it sends a *finish* message over e.
  $\star$ If j is *passive* then i becomes j's owner and j marks e as *owner-link*$_j$. It sends an *accept* message over e and changes its state to *captured*.
  $\star$ If j is in the first phase then j sends a *proceed* message over e.
- If j is *captured* then it checks whether its owner has finished the first phase. For

this purpose, it sends a *check* message over *owner-link*$_j$ (if it has already sent a check message, it waits for the reply to avoid congestion). If the owner replies that it has finished the first phase, then it sends a *finish* message to i and to any other node from which it has received (or will receive in the future) a *first-phase* message in the meantime. If the owner has not finished the first phase then j sends a *proceed* message to i and also to any other node from which it may have received a message in the meantime.

After node i has received responses to all k *first-phase* messages, it behaves as follows: It exits the first phase. If it has received a *finish* message then it does not enter the second phase and changes status to *killed*. Otherwise, it enters the second phase. It also updates its level number to the number of *accept* messages received in the first phase. In the second phase, node i tries to reach level k. For this purpose, it sends a *capture*($level_i, i$) message on each edge on which it received a *proceed* message. The rules for capturing are the same as in protocol $\mathcal{E}$ with the following changes: Nodes which have not started the second phase are regarded as *passive* by these capture messages. A node increases its level number only after receiving an *accept* response to each *capture* message sent in the second phase. After finishing the second phase, a node executes $\mathcal{F}$.

In $\mathcal{G}$, a base node finishes its first phase within 5 time units of waking up [Si92]. Furthermore, there will be a node which enters the second phase and a node which finishes the second phase to participate in $\mathcal{F}$. Hence, the protocol will elect a leader. Using the technique in [BKWZ87], we also extend our protocol to obtain a protocol resilient to f initial site failures, where $f < N/2$, which requires $O(Nf + NlogN)$ messages and $O(N/logN)$ time.

Lemma 4.3 *The time complexity of* $\mathcal{G}$ *is* $O(N/k)$.

Proof: A base node will finish its first phase within 5 time units of waking up. If node i wakes up spontaneously at time t then for i to participate in the second phase, each of its *first-phase* messages must go to a node which is in its first phase (this node must have awakened spontaneously after time $t-5$, otherwise it will have finished its first phase by time t) or is *passive* (this node will wake up by time $t+1$ as a result of i's message). Therefore, i is able to proceed to the second phase only if at least k nodes other than i wake up in the interval $[t-5, t+1]$.

Consider an interval of 5 time units, say $[m, m+5]$ where $m \geq 0$, during the execution of the protocol. We have the following cases:
(1) At least $k+1$ nodes wake up in the interval $[m-5, m+6]$.
(2) Less than $k+1$ nodes wake up in the interval $[m-5, m+6]$. In this case, we will show that some node will reach level at least k. All nodes that wake up before time m will finish their first phase by time $m+5$. Any node which completes its first phase in the interval $[m+5, m+6]$ will not be able to participate in $\mathcal{F}$ as a candidate (since it must have awakened in the interval $[m, m+6]$ and less than $k+1$ nodes wake up in the interval $[m-5, m+6]$). If a node has not already reached level k then let i be the node with the highest identity among the nodes which are in the second phase at time $m+5$. Let a *capture* message from i reach a node j which is not captured. We have the following cases:
(a) If j has not started the second phase then it will respond with an *accept* message.
(b) If j has started the second phase then it must be the case that j entered the second phase at or before time $m+5$ and therefore $j < i$ (by assumption). Hence, j will respond with an *accept* message.

If j is a captured node then it will send forward the message to its owner. If the $owner_j$ has not started second phase then it will send an *accept* message. Otherwise, we will show that $owner_j$ must have entered the second phase before time $m+5$. Assume not. Since nodes that wake up before time m enter the second phase before time $m+5$ and nodes that wake in the interval $[m, m+5]$ do not participate in the second phase, $owner_j$ must have awakened after time $m+5$. In this case, the *first-phase* message from $owner_j$ will reach j after time $m+5$. However, by this time, i would already have sent its *first-phase* message to j and therefore, $owner_j$ cannot capture j. Hence, $owner_j$ must have finished the second phase before time $m+5$ in which case it will send an *accept* message to i since $owner_j < i$ by assumption. Thus, i will receive all *accept* responses and therefore i will finish its second phase.
Hence, in each interval of 11 time units ($m-5$ to $m+6$), either at least k nodes wake up or some node will reach level k. Therefore, by time $11N/k$, either all nodes will be awake or some node will have reached level k. Then, from Lemma 4.1 and Lemma 4.2, the protocol will terminate in $O(N/k)$ time units. □

In [Si92], we show that the time complexity of $\mathcal{G}$ depends of the number of candidate nodes. By using the capturing pattern of the synchronous protocol in [AG85], we have obtained a message optimal protocol which requires $O(logN + min(r, N/logN))$, where r is the number of candidate nodes.

5 A Lower Bound

We will now prove that $\Omega(N/logN)$ is a lower bound on the time complexity of any message optimal protocol for leader election in complete asynchronous networks. We will restrict ourselves to *comparison-based* leader election algorithms. We prove the following theorem:

Theorem 5.1 *Any comparison-based protocol for leader election in a complete asynchronous network which sends less than Nd messages will require at least $N/16d$ time.*

Corollary 5.1 *Any message optimal protocol for leader election in a complete asynchronous network, i.e., requiring $O(NlogN)$ messages, will require $\Omega(N/logN)$ time.*

Proof of Theorem 5.1: For simplicity, assume that nodes have identities belonging to the set $\{1, \ldots, N\}$ and let $k = 2d$. A one-to-one function f from a set of processor identities to another set of processor identities is *order-preserving* if $i \leq j$ implies $f(i) \leq f(j)$. Two lists $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_m\}$ are *order-equivalent* if $(x_i \leq x_j) \Leftrightarrow (y_i \leq y_j)$. Following [FL87], we assume that each processor's local state consists of its identity, its initial state and the history of messages it has received so far. Further, a node sends its entire local state in each message. Intuitively, a process state includes all events that can potentially affect this state [Lam78] and the *happens before* ordering information between these events. Let *event*(i,t) denote the set of events which can potentially affect the state of i at time t in an execution. We say that *event*(i,t) and *event*(j,t') are *order-equivalent* if there exists an order-preserving function which maps *event*(i,t) to *event*(j,t') such that the happens-before relation is preserved. If *event*(i,t) and *event*(j,t') are order-equivalent then we say that the state of i at time t and state of j at time t' are order-equivalent. A comparison-based protocol cannot distinguish between order-equivalent states. Our proof involves showing that we can keep processes in order-equivalent states for a long period of time.

Let A be a protocol for leader election which requires Nd messages. Let Ex be the set of executions of A in which (1) all nodes wake up spontaneously at the same time, (2) all messages take ϵ time to reach their destination, where $\epsilon < 1/2$ and (3) if a set of messages arrive at the same time at a site then the messages are accepted in the increasing order of sender identities. Let Up_i denote the ordered list of edges from i to nodes $i+1,\ldots,i+k$, arranged in the increasing order of sites identities and $Down_i$ denote the set of edges from i to nodes with identities i–$1,\ldots,i$–k. Since a node cannot distinguish between untraversed incident links, the adversary has the freedom to choose any untraversed edge whenever the node wants to send a message over an untraversed edge. In particular, the adversary acts as follows: Whenever a node i has to send a message over a new edge, the adversary selects the edges first from Up_i. If all edges in Up_i have been used then it selects other unused edges. The actions of the adversary try to impose a symmetry on the nodes. We partition the nodes into the following sets: $\{S_1,\ldots,S_{N/k}\}$, where $S_i = \{k(i-1)+1,\ldots,ki\}$. Let $R = S_2 \cup \ldots \cup S_{m-1}$, where $m = N/k$ and $R' = S_1 \cup S_m$. As long as nodes in R remain in order-equivalent states, each node i will only communicate with nodes in $Up_i \cup Down_i$; otherwise each node in R will send messages over at least $k+1$ edges and the number of messages will exceed Nd. This fact and conditions (1)-(3) impose a symmetry on nodes in R. Intuitively, nodes in R cannot break symmetry without communicating with nodes in R'. Each node i in R executes the same protocol and communicates with at most k nodes with larger identities (belonging to Up_i) and at most k nodes with smaller identities (belonging to $Down_i$). However, nodes in R' are asymmetric with respect to these nodes. In any execution, let $nodes(i,t)$ denote the set of sites at which events in $event(i,t)$ occur. Then, for example, at time ϵ, sites 1 and i, where $i \in R$, may not be in order-equivalent states since 1 may receive a message from a node with a higher identity while i only receives messages from nodes with lower identities. However, at this time, $event(i,\epsilon)$ and $event(j,\epsilon)$, where $i,j \in R$, are order-equivalent. Hence, any node i in R will only send messages to nodes in $Up_i \cup Down_i$ at time ϵ. Observe that $nodes(i,\epsilon) \subseteq S_{y-1} \cup S_y \cup S_{y+1}$, where $i \in S_y$. If 1 sends a message at time ϵ to site $i \in S_2$ then it can force i and another site $j \in R$ to be in order-inequivalent states at time 2ϵ. However, for each node $i \in S_3 \cup \cdots \cup S_{m-2}$, if $nodes(i,2\epsilon) \subseteq S_{y-2} \cup \cdots \cup S_{y+2}$ then nodes in $S_3 \cup \cdots \cup S_{m-2}$ will be in order-equivalent states at time 2ϵ (in this case, no node would have received a message at time 2ϵ from a node in R' which, at time ϵ, was in a state order-inequivalent with respect to states of nodes in R). In general, we prove the following: Let $M_x = S_{x+1} \cup \cdots \cup S_{m-x}$ and $depth(i,t)$ denote the longest chain of messages involving events in $event(i,t)$.

Lemma 5.1 *There exists an execution in Ex such that at any time t and $i \in S_y \cap M_x$, where $x \leq m/4$, if $depth(i,t) \leq x$ and $nodes(i,t) \subseteq S_{y-x} \cup \cdots \cup S_{y+x}$, then for all $j \in M_x$, $event(i,t)$ and $event(j,t)$ are order-equivalent.*

Proof Outline: We prove this by induction on x. The result is immediate for $x = 0$ since processes are initially in order-equivalent states. Assume that the hypothesis holds for $x \leq l$. Then there exists an execution in Ex such that at any time t, if $depth(i,t) \leq l$ and $nodes(i,t) \subseteq S_{y-l} \cup \cdots \cup S_{y+l}$ then for all $j \in M_l$, $event(i,t)$ and $event(j,t)$ are order-equivalent. Let t be the maximum time in this execution at which $depth(i,t) \leq l$ for $i \in M_l$ (*i.e.*, any message sent after this time increases the depth). Assume that at some time t' in this execution, $depth(i,t') = l+1$ and $nodes(i,t') \subseteq S_{y-l-1} \cup \cdots \cup S_{y+l+1}$, where $i \in M_{l+1}$. Let $j \in M_{l+1}$. Since $i,j \in M_l$, $event(i,t)$ and $event(j,t)$ are order-equivalent. Let p send a message to i at time t which is in $event(i,t')$ but not in $event(i,t)$. Then $depth(p,t) = l$ and

$p \in M_l$ (otherwise, we can show a contradiction since $i \notin Up_p \cup Down_p$). Let $q = j-(i-p)$. Then $q \in M_l$. From the induction hypothesis, $event(p,t)$ and $event(q,t)$ are order-equivalent. Hence, q will also send a message to j. Therefore, $event(i,t')$ and $event(j,t')$ will be order-equivalent. □

Lemma 5.2 *There exists an execution in Ex such that at any time t and $i \in M_{m/4} \cap S_y$, if $nodes(i,t) \subseteq S_{y-m/4} \cup \cdots \cup S_{y+m/4}$ and $depth(i,t) \leq m/4$ then i must have communicated only with nodes in Up_i and $Down_i$.*

Proof: Assume not. Assume that i sends a message to a node not in $Up_i \cup Down_i$. Let $j \in M_{m/4}$. Then by Lemma 5.1, there exists an execution in which $event(i,t)$ and $event(j,t)$ are order-equivalent. Since i sends a message to a node not in $Up_i \cup Down_i$, j will also send a message to a node not in $Up_j \cup Down_j$ (since order-equivalent states are indistinguishable to a comparison-based protocol). Thus, each node in $M_{m/4}$ will send at least $k+1$ messages. Since $|M_{m/4}| = N/2$, the execution will involve at least $N(k+1)/2$ messages which is a contradiction. □

We will make use of symmetry between nodes in R to construct an execution in which nodes in R' wake up much later in the protocol to break symmetry. Let ex be an execution of A. Let the nodes be partitioned into three sets, P_1, P_2 and P_3 such that $P_1 = \{1,\ldots,p_1\}$, $P_2 = \{p_1+1,\ldots,p_2\}$ and $P_3 = \{p_2+1,\ldots,N\}$. Assume that (1) nodes in P_1 and P_3 wake up at time t, (2) no messages have been sent by nodes in P_2 to nodes in P_1 and P_3 up to time t and (3) all messages sent at or after time t take ϵ time units. Let $g(ex,P_2)$ denote the execution in which all nodes in P_2 wake up $1-\epsilon$ time earlier than in ex but all links incident on nodes in P_2 remain idle in the interval $[t-(1-\epsilon),t]$ *i.e.*, transmission of messages does not make progress during this period. Due to the asynchronous nature of the network, no node can distinguish between ex and $g(ex,P_2)$. This technique of increasing message transmission time without violating the happens before ordering is similar to the one in [AFL83].

Let M be the set of messages in ex sent at time t from i to j, where $i,j \in P_2$ and j receives no messages from nodes in $P_1 \cup P_2$ at time $t+\epsilon$. Let $h(ex,P_2)$ denote the execution in which links incident on nodes in P_2 are not idle in the period $[t-(1-\epsilon),t]$ but all messages sent in this interval except those in M take $\epsilon+(1-\epsilon)$ time. The effect of this transformation is to increase the delays on the links from ϵ to $\epsilon+1-\epsilon$, which cannot be distinguished from links remaining idle for $1-\epsilon$ time units. Hence, ex and $h(ex,P_2)$ are indistinguishable to all nodes.

We will construct a set of executions in a series of steps which takes $O(N/k)$ time. Let ex be an execution in Ex which satisfies Lemma 5.2. Let $A_1 = S_1 \cup \cdots \cup S_{m/2-1}$, $B_1 = S_{m/2}$ and $C_1 = S_{m/2+1} \cup \cdots \cup S_m$, where $m = N/k$. All nodes wake up at time $t = 0$ and no messages are sent before that time. Furthermore, all messages take ϵ time units. Hence, ex and $h(ex,B_1)$ are indistinguishable to all nodes. Let $ex_1 = h(ex,B_1)$. Let $A_2 = S_1 \cup \cdots \cup S_{m/2-2}$, $B_2 = S_{m/2-1} \cup S_m \cup S_{m/2+1}$ and $C_2 = S_{m/2+2} \cup \cdots \cup S_m$. In ex_1, no messages are sent to nodes in A_2 and C_2 before time $1-\epsilon$ (since nodes in $S_{m/2}$ communicate only with nodes in $S_{m/2-1}$ and $S_{m/2+1}$ until nodes not in R' wake up (From Lemma 5.2)). Further, all messages sent at or after time $1-\epsilon$ take ϵ time units. Therefore, ex_1 and $h(ex_1,B_2)$ are indistinguishable to all nodes. Since ex_1 and ex are indistinguishable, ex and $h(ex_1,B_2)$ are also indistinguishable to all nodes. Let $ex_2 = h(ex_1,B_2)$. Continuing in this way, we can construct an execution $ex_{m/4}$, where $A_{m/4} = S_1 \cup \cdots \cup S_{m/4}$, $B_{m/4} = S_{m/4+1} \cup \ldots \cup S_{3m/4}$ and $C_{m/4+1} = S_{3m/4} \cup \cdots \cup S_m$, which is indistinguishable from ex. The execution time of $ex_{m/4}$ is at least $m/4(1-\epsilon)$. Since $m/4(1-\epsilon) = N/4k(1-\epsilon) = N/8d(1-\epsilon) \geq N/8d(1/2) = N/16d$, we have an execution which requires $N/16d$ time. □

6 Conclusion

In this paper, we have presented distributed algorithms for leader election in complete networks. For asynchronous networks with a sense of direction, we first presented a simple protocol which requires $O(N)$ messages and $O(\sqrt{N})$ time. We then improved the time complexity of this protocol to $O(logN)$ time. An interesting question is whether synchronized clocks can be used to improve the time complexity of this protocol.

For completer asynchronous networks without sense of direction, we also showed a lower bound of $\Omega(N/logN)$ on the time complexity of any message optimal leader election protocol. We presented a protocol which requires $O(N/logN)$ time and $O(NlogN)$ messages. In [AG85], a lower bound of $\Omega(logN)$ on the time complexity of any message optimal protocol for synchronous complete networks was shown. This proves that introducing asynchrony may result in a loss in speed by a factor of $N/(logN)^2$. In [Si92], we study the problem of leader election in partially synchronous networks and present lower bounds for such networks.

References

[AFL83] Arjomandi, E., Fischer, M., and Lynch, N. Efficiency of synchronous versus asynchronous distributed systems. *Journal of the ACM*, 30(3), 1983.

[AG85] Afek, Y. and Gafni, E. Time and message bounds for election in synchronous and asynchronous complete networks. In *Proceedings of ACM Sym. on Principles of Distributed Computing*, 1985.

[ALSZ89] Attiya, H., Leeuwen, J., Santoro, N., and Zaks, S. Efficient elections in chordal ring networks. *Algorithmica*, 4, 1989.

[BKWZ87] Bar-Yehuda, R., Kutten, S., Wolfstahl, Y., and Zaks, S. Making distributed spanning tree algorithms fault-resilient. In *STACS*, 1987.

[FL87] Frederickson, G. and Lynch, N. The impact of synchronous communication on the problem of electing a leader in a ring. In *Proceedings of ACM Sym. on Theory of Computing*, 1987.

[KMZ84] Korach, E., Moran, S., and Zaks, S. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In *Proceedings of ACM Sym. on Principles of Distributed Computing*, 1984.

[Lam78] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communciation of the ACM*, 21(7), July 1978.

[LMW86] Loui, M., Matsushita, T., and West, D. Election in complete networks with a sense of direction. *Info. Processing Letters*, 22, April 1986.

[Si91] Singh, G. Efficient distributed algorithms for leader election. In *IEEE International Conference on Distributed Computing Systems*, 1991.

[Si92] Singh, G. Upper and lower bounds for leader election in complete networks. In *Technical Report 92-8, Kansas State University*, 1992.