Elections in Anonymous Networks

Yehuda Afek

Computer Science Department
Tel Aviv University
Ramat-Aviv 69978
Israel

Yossi Matias*

Computer Science Department
Tel Aviv University, Israel
and
Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract

We address the problem of electing a leader in an anonymous, asynchronous network of arbitrary topology. Our algorithms are considerably simpler than known algorithms and have equal or improved communication complexity.

1 Introduction

Consider the problem of electing a leader in an anonymous, asynchronous network of arbitrary topology. Angluin (1980) showed that from symmetry considerations, there is no deterministic algorithm to elect a leader (i.e., to break the symmetry) in a general anonymous network. Following (Angluin, 1980) many probabilistic algorithms for electing a leader, and/or breaking the symmetry were proposed (Abrahamson *et al.*, 1986; Attiya *et al.*, 1988; Cole and Vishkin, 1989; Itai and Rodeh, 1990; Frederickson and Santoro, 1986; Schieber and Snir, 1989). However, only Schieber and Snir (1989) have considered the leader election problem for an arbitrary topology asynchronous network.

When discussing election algorithms for anonymous networks one has to consider the following two factors: *knowledge of the network size*, and *termination detection* (Attiya *et*

*The research of this author was partially supported by NSF grant NSF-CCR-8906949.

al., 1988; Schieber and Snir, 1989). In (Itai and Rodeh, 1990) it was shown that a leader can be randomly elected in a ring, with termination detection, and with fixed error probability $\epsilon < 1$, only if an upper bound is known on the network size. In case of an error more than one leader is elected in the network. Furthermore, Itai and Rodeh continued to show that the problem can be solved with termination detection and without error only if the ring size is known up to a factor of two; however, only the expected complexities are bounded. Schieber and Snir (1989) extended all of these results to an arbitrary network, under a variety of assumptions on the processors *a priori* knowledge of the network. They have presented efficient algorithms for constructing a spanning tree and electing a leader (see Figure 1 for summary of their results).

In this paper we present three types of algorithms. In Section 2 we give a simple election algorithm without termination detection for the case that nothing is known about the network size n. This algorithm may err with probability $< \epsilon$ by electing more than one leader. In Section 3 we show that when a lower bound is given on the network size the algorithm of Section 2 can be made more efficient (still without termination detection). The reduced complexity is in the expense of having a new type of error: electing no leader (still with probability $\leq \epsilon$). In Section 4 we turn to the design of election algorithms with termination detection when both lower and upper bounds are given. Finally, we show in Section 5 that for the problem of electing a leader, complete anonymous networks are more powerful than arbitrary networks. Specifically, we show that the complexity of probabilistically electing a leader in an anonymous complete network is the same as in an anonymous ring, the cost in both topologies being lower than in dense networks of arbitrary topology. (The same relationships hold in the case of deterministic election algorithms without anonymity.) A summary of the results is given in Figure 1.

Aside from being considerably simpler, the results in this paper improve the time complexity and the bit communication complexity of Schieber and Snir (1989). Schieber and Snir suggest a way to reduce the message complexity by increasing the message size. Contrary to this approach the algorithms presented here have increased message complexity and reduced message size, which result in an overall decrease in the bit complexity for sparse networks. Furthermore, the approach taken here leads to algorithms with better space complexity. The algorithms in Sections 2 and 3 require at most $O(\log\log n + \log 1/\epsilon)$ bits of state information per node (it is always equal to the message size), while in (Schieber and Snir, 1989) the algorithms require $O(\log n + \log 1/\epsilon)$ bits of state information per link. As in (Schieber and Snir, 1989) all our algorithms construct a spanning tree which can then be used for many applications, such as counting the number of nodes, assigning unique *id*'s to the nodes, constructing a minimum weight spanning tree, etc.

An important contribution of our paper is the distributed randomized *id*s selection process; this is at the heart of all the algorithms presented and would probably have other applications. In all the algorithms each node selects an *id* at random. The *id*s selection procedure is designed such that two conditions are simultaneously satisfied with high prob-

Election	Known about n	Term. Detect.	Message Complexity*	Message Size	Time
[SS-89]	nothing	w/o	$m \cdot \log(n^2/m)$ (worst case)	$\log n$	n (w.c.)
[SS-89]	nothing	w/o	$m \cdot \log\log n + n \log n$ $(m \cdot f(n) + n \log n)$	$\log n$ $(\log F(n))$	$n \log\log n$ $nf(n)$
§2	nothing	w/o	$m \log n$	$\log\log n$	D
§3	$L \leq n$	w/o	$m \cdot \log(n/L)$	$\log\log(n/L)$	D
§3	$L < n \leq kL$ $k \geq 1$	w/o	$m \cdot \log k$ (worst case)	$\log\log k$	D (w.c.)
[SS-89]	$n \leq U$	with	$n \cdot \log n + m$ (worst case)	$\log U$	n (w.c.)
§3	$n \leq U$	with	$n \cdot \log n + m$ (worst case)	$\log U$	n (w.c.)
§4 Thm 2	$L < n \leq 2L$	with	m (≤ 1 leaders)	$\log\log n$ or $\log n$†	n (w.c.)
§4 Thm 3	$L < n \leq kL$ $k \geq 1$	with	$m \cdot k \log k$ (worst case)	$\log\log n$ or $\log n$†	n (w.c.)
§5	complete network	with	n (worst case)	$\log n$	$\log n$

* The message complexity is the minimum between the stated complexity and $O(mD)$; m is the total number of links; D is the network diameter; the error probability ϵ is fixed; w.c. = worst case; w/o = without; $F(n) = \max\{i : f(i) = f(n)\}$. † The expected number of messages of size $\log n$ is $O(n)$.

Figure 1: Summary of our results and the corresponding results from (Schieber and Snir, 1989) ("[SS-89]").

ability: (a) the maximum id is unique; and (b) the number of different ids selected is small ($O(\log n)$).

2 Networks with unknown size

In this section we present a simple probabilistic algorithm for leader election in an asynchronous network of processors. The processors are assumed to be anonymous and have no knowledge of the network size or topology. The only thing known to each processor is its collection of incoming and outgoing ports by which it receives and sends messages from and to its neighbors. The algorithm succeeds with probability $1-\epsilon$. That is, with probability $1-\epsilon$, a unique leader is elected and otherwise several leaders may be elected. However, it is never the case that no leader is elected. Throughout this paper we let $r = 1/\epsilon$.

The algorithm, denoted $ELECT$, is shown in Figure 2. It proceeds in two steps. First, each node randomly selects an id, and second each node tries to broadcast its id over the network. Upon receiving an id a node forwards it only if it is larger than any id the node has seen so far. Thus, eventually the largest id captures the entire network.

The crux of the algorithm is to select the ids in such a way that the following two conditions are satisfied with high probability: (a) The maximum id selected would be unique, thus a single node is distinguished from the rest; (b) The number of different ids is small. (The id selection procedure distributively estimates the size of the network without explicitly revealing this information to any single node.)

More precisely, each processor P_i randomly selects two numbers t_i and s_i as follows: (1) It tosses a fair coin until it gets a $Head$ for the first time. Let t_i be the number of these tosses. (2) The second number, s_i, is randomly selected in the range $[1..d]$, where $d = O(r \log r)$. (The exact value of d will be given in the analysis.) Let id_i, the id of processor P_i, be the ordered pair $< t_i, s_i >$. A lexicographic order on the ids is naturally defined. Denote this procedure of choosing $id_i =< t_i, s_i >$ as $Choose(id_i)$.

2.1 Analysis

Let n be number of network nodes, m the number of network links, D the diameter of the network and $r = 1/\epsilon$. In this subsection we prove

Theorem 1 *Algorithm $ELECT$ will eventually reach quiescence. Upon reaching quiescence a unique leader is elected with probability $\geq 1-\epsilon$. The expected message complexity of $ELECT$ is $O(mr \log n \log r)$, and the expected size of each message is $O(\log\log n + \log r)$ bits. The time complexity is $O(D)$. All complexities are with probability $> 1 - n^{-\alpha}$ for any constant $\alpha > 0$.*

The following straightforward claim is given here without a proof:

Claim 1 *Eventually, algorithm $ELECT$ will reach quiescence at which time max_i is the same at all nodes.*

Following the above it is readily seen that the worst case message complexity of the algorithm is $O(mn)$. Let $M = \max\{id_i : i = 1, \ldots, n\}$ and let $parent_i$ be the link over which node i has received M for the first time (if $id_i = M$ then $parent_i := nil$). We have

Procedure $Initialize$ /* performed upon waking up */
call $Choose(< t_i, s_i >)$; /* P_i randomly selects $id_i =< t_i, s_i >$ */
$max_i := id_i \; :=< t_i, s_i >$; $leader$:= true;
send id_i to all neighbors. /* P_i broadcasts its id */

Upon receiving id **at node** i **over link** l**:**
if id_i not defined then call $Initialize$;
if $id \leq max_i$
then ignore /* P_i purges this message */
else $max_i := id$; $leader$:= false;
send id to all neighbors. /* P_i broadcasts the new max_i */

Procedure $Choose(id)$ /* selecting an id for the node */
$j := 1$;
while fair_coin = $tail$ do $j := j + 1$;
s := randomly selected from $[1, \ldots, d]$; /* where $d = 36r \ln(4r)$ and $r = 1/\epsilon$ */
return (id:=$< j, s >$) ;

Figure 2: Algorithm ELECT at P_i

Claim 2 *If there is a unique node with* $id_i = M$ *then the set of links* $\{parent_i : i = 1, \ldots, n\}$ *produced by algorithm ELECT is a spanning tree of* G.

Proof: Let T' be the directed graph that consists of the pointers $parent_i$ and T be the corresponding undirected graph. Clearly, T spans the nodes of G. T is connected, since each node can reach the leader by following the pointers $parent_i$. The number of edges in T is $n - 1$ since each node except the leader has exactly one $parent_i$. Therefore, T is a rooted spanning tree of G. ∎

Let $t = \max\{t_i : i = 1, \ldots, n\}$.

Claim 3 *(i)* $Prob(t \leq \log(rn)) \geq 1 - \epsilon$*; (ii)* $Prob(t \leq (\alpha + 1) \log n) \geq 1 - n^{-\alpha}$ *for any* $\alpha > 0$*; and (iii)* $E[t] \leq 2(\lceil \log n \rceil + 1)$.

Proof: (i) For any fixed i, Prob $(t_i > \log(rn)) = (1/2)^{\log(rn)} = (1/rn)$. Therefore, Prob $(\exists i \in \{1..n\}$ s.t. $t_i > \log(rn)) \leq 1/r = \epsilon$.

(ii) Let $\epsilon = n^{-\alpha}$ then $\log(rn) = \log n^{\alpha+1} = (\alpha + 1) \log n$.

(iii) Let $N_i = |\{j : t_j > i\}|$. Consider the following series of Bernoulli trials: A *success* in the i'th trial is defined to be the case that $N_{i+1} \leq \min(\frac{N_i}{2}, 0)$. It is clear from Procedure

$Choose(<t,s>)$ that Prob($success$) $\geq \frac{1}{2}$. Let x be the number of trials until the $(\lceil \log n \rceil + 1)$'st successful trial. Clearly, t is upper bounded by x and therefore $E[t] \leq E[x] = 2(\lceil \log n \rceil + 1)$. ∎

Claim 4 *(i) With probability* $\geq 1 - n^{-\alpha}$ *(for any* $\alpha > 0$*), the message complexity of algorithm* $ELECT$ *is* $O(m \log n \cdot \alpha r \log r)$*; (ii) The expected message complexity of algorithm* $ELECT$ *is* $O(m \cdot \log n \cdot r \log r)$*.*

Proof: (i) Let $M = \max\{id_i : i = 1..n\}$. Recall that $id_i = <t_i, s_i>$ and that $s_i = O(r \log r)$. Thus, by Claim 3(ii), we have $M = O(\alpha r \log r \log n)$ with probability $\geq 1 - n^{-\alpha}$. In algorithm $ELECT$, node P_i sends id only if id is larger than max_i. Thus, through each link no value is sent twice and the number of messages sent through a link is at most M. The total number of messages is at most mM which is $O(\alpha r \log r \cdot m \log n)$ with probability $\geq 1 - n^{-\alpha}$.

(ii) Follows from the proof of (i) and Claim 3(iii). ∎

Claim 5 *(i) The size of each message is* $O(\log \log n + \log r + \log \alpha)$ *bits with probability* $\geq 1 - n^{-\alpha}$*; (ii) The expected size of each message is* $O(\log \log n + \log r)$ *bits.*

Proof: Each message is of size $O(\log M)$. The claim follows from Claim 3. ∎

Following Claims 4 and 5, the bit complexity of algorithm $ELECT$ is, with probability $\geq 1 - n^{-\alpha}$, $O(m \log n \cdot \alpha r \log r(\log \log n + \log r + \log \alpha))$, and the expected bit complexity is $O(m \log n(\log \log n + \log r) \cdot r \log r)$. The space complexity of Algorithm $ELECT$ is $O(\log \log n + \log r + \log \alpha)$ bits per node, with probability $\geq 1 - n^{-\alpha}$, and the expected space complexity is $O(\log \log n + \log r)$ bits per node.

The main idea in algorithm $ELECT$ is to design Procedure $Choose(<t,s>)$ such that with high probability there is exactly one node whose $id = <t,s>$ is larger than all the other ids. The following lemma proves the existence, with high probability, of such a node.

Lemma 6 *There is a unique node* P_i *with* $id_i = M$*, with probability* $\geq 1 - \epsilon$*.*

Proof: Denote as *candidate* a node i such that $t_i \geq thresh = \log n - h$ for some parameter h that will be fixed later. Intuitively, $\log n - h$ is a threshold that identifies nodes with "high" ids. The leader will be the candidate with the highest id.

We prove the lemma in two steps. We first show (Claim 9) that (a) there are $\Theta(\log r)$ candidates, with probability $> 1 - \epsilon/2$. This ensures that, with high enough probability, there is at least one candidate but not too many. We then show (Corollary 11) that (b) with probability $> 1 - \epsilon/2$, $\max\{s_i : P_i \text{is candidate with highest} t_i\}$ was chosen by only one candidate. This will suffice to prove that there is a unique node with $\max\{id_i\}$ with probability $\geq 1 - \epsilon$.

To prove Claim 9 we need the following:

Lemma 7 (Chernoff, 1952; Angluin and Valiant, 1979)

If X is binomial with parameters (n, p) then for all $0 < b < 1$

(1) Prob $(X \geq \lceil (1+b)np \rceil) \leq \exp(-b^2 np/3)$

(2) Prob $(X \leq \lfloor (1-b)np \rfloor) \leq \exp(-b^2 np/2)$.

Corollary 8 *If X is binomial with expectation c then*

(1) Prob $(X \geq 3c/2) \leq \exp(-c/12)$

(2) Prob $(X \leq c/2) \leq \exp(-c/8)$.

Proof: Let $b = 1/2$ and $np = c$ in Lemma 7. ∎

Recall that a *candidate* is a node i whose $t_i \geq \log n - h$ for some parameter h. Let X be the number of candidates. The probability that a node is a candidate is $2^{h-\log n} = 2^h/n$ (= the probability to get $\log n - h$ *Tails* in $\log n - h$ coin tosses). As the number of nodes is n, we have $E[X] = 2^h$. Let $c = E[X] = 2^h$.

In the following claim we show that for a particular value of c the number of candidates is $\Theta(\log r)$ with probability $> 1 - \epsilon/2$.

Claim 9 *Let $c = 12\ln(4r)$, then Prob $(c/2 \leq X \leq 3c/2) > 1 - \epsilon/2$.*

Proof: By Corollary 8 we have Prob $(c/2 \leq X \leq 3c/2) \geq 1 - [\exp(-c/12) + \exp(-c/8)] = 1 - [\exp(-\ln(4r)) + \exp(-\frac{3}{2}\ln(4r))] > 1 - 2/(4r) = 1 - \epsilon/2$. ∎

This concludes the proof of (a) in the proof of Lemma 6. It remains to show that among the candidates, the maximum is unique with high probability. We first show the following general claim:

Claim 10 *Let p be the probability that in k drawings from a domain of size d, the largest label (among those that were drawn) is drawn only once. Then, $p \geq 1 - (k/d)$.*

Proof: Let Q be the following process of k restricted drawings from a domain of size d, in which it is guaranteed that the largest label is drawn only once. In Q the $(i+1)$'st drawing is restricted to be from all labels except for the maximum drawn label over the first i drawings. Let K_Q be the number of different sequences of labels that could be drawn by Q. By a simple counting argument, $K_Q = d(d-1)^{k-1} > (d-1)^k$.

Let K be the number of possible different results of a series of k (unrestricted) drawings from a domain of size d, in which the largest label is drawn only once. Clearly, $K \geq K_Q$. Therefore, $p = K/d^k \geq K_Q/d^k > (d-1)^k/d^k = (1-1/d)^k > 1-(k/d)$. ∎

Corollary 11 *If the number of candidates is* $\leq g$ *and* s_i *was chosen from the range* $[1..2rg]$ *then* $\max_i\{id_i\}$ *is unique with probability* $> 1-\epsilon/2$.

Proof: We need to consider only the candidates with the highest t_i. Assume that all candidates have the same t_i. There are $\leq g$ drawings from a domain of size $2rg$. Claim 10 implies that $\max_i\{s_i : P_i \text{ is a candidate}\}$ is unique with probability $\geq 1-g/2rg = 1-\epsilon/2$. ∎

This concludes the proof of (b) in the proof of Lemma 6.

Let $g = 18\ln(4r)$ (so as to satisfy $g = 3c/2$ for $c = 12\ln(4r)$ in Claim 9). That is, the domain d from which s_i is chosen is $d = 2rg = 36r\ln(4r) = O(r\log r)$. By Claim 9 and Corollary 11, with probability $\geq 1-\epsilon$, there is a unique node P_j with $id_j = \max\{id_i : i = 1,\ldots,n\}$; this concludes the proof of Lemma 6. ∎

Finally, we determine the time complexity of Algorithm $ELECT$.

Claim 12 *The time complexity of algorithm* $ELECT$ *is* $O(D)$, *where* D *is the diameter of the network.*

Proof: First note that within D units of time all nodes are awakened. Then with an additional D units of time the algorithm must reach quiescence since the time complexity is dominated by the time it takes M, the largest value among the ids, to reach each node. This time is therefore $O(D)$. ∎

The above analysis is summarized in Theorem 1.

Leader election on a ring: As a corollary a leader can be elected on a ring of unknown size with error probability ϵ, with $O(n\log n \cdot r\log r)$ messages, each of size $O(\log\log n + \log r)$. Thus, the bit complexity is $O(n\log n\log\log n)$ (for fixed ϵ), an improvement over the $O(n\log^2 n)$ bound of (Schieber and Snir, 1989) and of (Itai and Rodeh, 1990). (Abrahamson *et al.* (1989) present an $O(n\log n)$ expected bit complexity algorithm by assuming either that the ring size is known, or that the nodes have unique ids.)

2.2 A different view of the algorithm

In Procedure $Choose(<t, s>)$ each node selects two random numbers t_i and s_i. The two can be combined into one number by tossing a biased coin to select t_i and eliminating s_i. More precisely, the probability of $Head$ in each coin toss is $p = O(\epsilon)$.

We can look at the process of choosing id's in a more general setting. We actually want each node to select a value j, according to some probability function p_j. We chose p_j to be $p_j = (1-p)^{j-1}p$. A natural question is whether one can find another p_j that will result in better complexities. More precisely, we look for a probability function p_j that guarantees that the maximum value that was selected by a node is unique with high probability, and that the number of different values that were selected is minimal. In our case, the number of different values is $O(\log n)$.

The following claim (Alon, 1989) indicates that our choice is optimal.

Claim 13 *Let p_j be a probability function for which the maximum value selected by any node is unique (i.e., it was selected by only one node) with probability $\geq 1-\epsilon$. Then, there are infinitely many n's for which the expected number of different values that are selected by n nodes is $\Omega(\log n)$.*

3 Networks with known bounds on size

In the previous section we gave an election algorithm when nothing is known about n, the network size. In the following sections we add stronger and stronger assumptions on what is known about lower and upper bounds of n. In this section the knowledge of a lower bound on n is used to reduce the complexities of the algorithm. The knowledge of an upper bound on n is used to guarantee that the complexities are worst case. The knowledge of both lower and upper bounds is used to achieve reduced complexities that are worst case. In Section 4, the knowledge of the network size to within a factor of 2 is used to derive an algorithm with termination detection.

Knowledge about lower bound: When a lower bound L on the network size is known, then a lower bound L_{id} on the maximum id is known as well, with high probability. Hence, each node whose selected id is smaller than L_{id} gives up its candidacy for leadership without trying to broadcast its id. This reduces the message complexity to $O(m \cdot r \log r \log(r\frac{n}{L}))$, with probability $\geq 1-\epsilon$. Moreover, to save in the message size, the remaining candidates broadcast only the difference between their id and L_{id}, which is $O(r \log r \log(r\frac{n}{L}))$, with probability $\geq 1-\epsilon$. Note that in the case that ϵ is fixed and $n = O(L)$, the message

Procedure $Initialize(i, L)$ /* performed upon waking up */ call $Choose(< t_i, s_i >)$; $t'_i := t_i - thresh$; /* $thresh = \log L - \log(12\ln(4r))$ */ $id_i := < t'_i, s_i >$; $max_i := id_i$; $leader :=$ true ; if $t_i > thresh$ then send id_i to all neighbors.

Figure 3: Modified Procedure *Initialize*

complexity is $O(m)$, each message of size $O(1)$ bits. The new initialization procedure is given in Figure 3.

Specifically, we claim that $L_t = \log n - h'$ is a lower bound on $t = \max_i\{t_i\}$, with probability $\geq 1-\epsilon$, for $h' = \log(\frac{n}{L}12\ln(4r))$. This is implied by Claim 9 and by the fact that $h' \geq h = \log(12\ln(4r))$ there. When evaluating L_t we get $L_t = \log L - \log(12\ln(4r))$ which is independent of n and thus every node knows L_t a priori (assuming that ϵ and L are given to all the nodes). Moreover, by Claim 3(i), $t - L_t \leq \log(rn) - L_t = \log(r\frac{n}{L}12\ln(4r)) = O(\log(r\frac{n}{L}))$, with probability $\geq 1-\epsilon$. Thus the message complexity and size are as stated.

The above reduction in the communication complexities is in the expense of introducing a new type of error, in which no leader is elected (with probability $< \epsilon/2$ by Claim 9). This happens when no message is sent by any node.

Knowledge about upper bound: If an upper bound U on n is known, ($n \leq U$), then all complexities are guaranteed to be worst case. Let $\bar{t} = \log(rU)$. Recall that in Algorithm $ELECT$, $\max_i\{t_i\} \leq \bar{t}$ with probability $\geq 1-\epsilon$ (Claim 3(i)). We modify Algorithm $ELECT$ so that if a node's t_i is larger than $\bar{t}$ then it is modified to be $\bar{t}$. Note that all analysis of uniqueness remains correct. We thus guarantee that the complexities are worst case.

Remark: If an upper bound $U \geq n$ on the size of the network is known then the problem can be easily reduced to the non-anonymous case as follows: Each node randomly selects an id from the range $[1..U^{2+\alpha}]$. With probability $1-n^{-\alpha}$, the ids are all distinct. A standard leader election algorithm (Gallager *et al.*, 1983; Awerbuch, 1987) for non-anonymous networks can be employed now to achieve $O(m + n\log n)$ messages of size $O(\log U)$ bits and $O(n)$ time algorithm. These algorithms are with termination detection. Note, however, that the standard algorithms were not designed to deal with the case that the ids are not distinct. Therefore, in case of failure to select distinct ids four undesirable events may occur: the algorithm may elect more than one leader, the algorithm's message complexity may grow without bound, the algorithm may enter a deadlock, and the algorithm may enter a state not specified by the code. The probability that multiple leaders are elected is less than $n^{-\alpha}$ for any constant α. To prevent the message complexity from growing without a bound the controller of Afek *et al.* (1987) may be employed, thus increasing the complexity to $O(n \log^3 n)$. One can check that the algorithms of Gallager *et al.* (1983) and Awerbuch (1987) do not enter a deadlock if the *id*s are not unique, however the code may get to unspecified states. A node detecting such a state initiates a broadcast of a *reset* message that causes all the nodes to start the algorithm all over. After receiving a reset message a node starts a new execution only after all its neighbors acknowledge receiving the reset message.

Knowledge about lower and upper bounds: When both an upper and a lower bounds on n are known, $L < n \leq U$, we employ the two modifications above simultaneously. That is, a node broadcasts its *id* only if its t_i is greater than L_t and it truncates its t_i to $\bar{t}$ if necessary. This results with an algorithm without termination detection whose worst case message complexity is $O(m \cdot r \log(r\frac{U}{L}) \log r)$, each message is of size $O(\log r + \log\log \frac{U}{L})$ bits. The algorithm succeeds with probability $\geq 1 - \epsilon$.

4 Leader election with termination detection

The algorithms presented in the previous sections elect a leader without termination detection. Itai and Rodeh (1990) showed that it is possible to elect a leader with termination detection in an anonymous network only if an upper bound $U > n$ on the network size is known. Schieber and Snir (1989) considered this case. They gave an election algorithm with termination detection that succeeds with probability $\geq 1 - \epsilon$. Its message complexity is $O(n \log n + m)$ messages (worst case), each message of size $O(\log U)$.

By *termination detection* we mean that if a leader is elected then eventually each node enters a *halt* state, after which the node stops running the algorithm, and ignores all messages that arrive afterwards. A successful run is one in which a unique leader is elected. A run of the algorithm in which no leader is elected (this happens with probability $< \epsilon$) is by definition without termination detection; otherwise, the algorithm may be re-run until a leader is elected. (The expected number of re-runs would be $1/\epsilon$.)

In this section we present a sequence of algorithms, each builds and improves on the previous one for the case that n is known within a factor of two, i.e., an L is given such that $L < n \leq 2L$. The algorithms are made efficient by letting them err so that no node is elected as a leader.

The first algorithm, called $ELECT2$, detects termination if and only if at least one leader is elected and it succeeds with probability $\geq 1 - \epsilon$; with probability ϵ no leader is elected (in which case there is no termination detection – see comment above). A description of the algorithm is given in Figures 4 and 5. Algorithm $ELECT2$ is a simple modification of algorithm $ELECT$ and uses $O(m)$ messages (for fixed ϵ), each of size $O(\log \log n)$, and $O(n)$ messages of size $O(\log n)$ each. For the case of a ring this algorithm improves upon the $O(n \log n)$ upper bound of Itai and Rodeh (1990) to an optimal algorithm with $\Theta(n)$ message complexity (for a fixed ϵ), thus matching the bit complexity of Abrahamson *et al.* (1989). This should be contrasted with the $\Omega(n \log n)$ lower bound for deterministic election in non-anonymous rings (Burns, 1980; Pachl, 1985) and with the $\Omega(n \log n)$ lower bound of Pachl (1985) for randomized leader election algorithms even for the case that all nodes have distinct ids.

Algorithm $ELECT2$**:** When algorithm $ELECT$ quiesces, max_i is the same at all the nodes, and the parent links of the nodes constitute a spanning forest of the network with one or more trees in it. The algorithm is successful in electing a leader if the forest is a single tree. If it is known that $L < n \leq 2L$ then the tree of the elected node must be of size greater than L. Moreover, if the size of a tree is greater than L then the root of that tree can safely assume the "leadership" among nodes with the same id since there can be only one such tree for each id in the network. Thus, to elect a leader with termination detection when n is known within a factor of two we provide the root of each tree in the algorithm the knowledge of its size. If its size is greater than L then the root assumes leadership and performs a broadcast and collate of $lead(id)$ messages, by which it attempts to capture the entire network. This will happen unless a larger id will be encountered.

To provide the root of each tree with its size the broadcast mechanism of $ELECT$ is replaced by a broadcast and collate (algorithm pif from (Segall, 1983); see also (Chang, 1979)) and the size of each tree is then collated in the collate phase. To facilitate this each node marks the link over which it receives the largest id for the first time as its $parent$ link, thus constructing a spanning forest.

When a root P_i receives acknowledgments from all neighbors, it also knows the size S_i of its tree. If $S_i \leq L$ then P_i knows that it is not the leader. It happens when P_i's tree is surrounded by other trees with the same id. If $S_i > L$ then P_i is still not necessarily the leader. Again, it is possible that its tree is surrounded by other trees with the same id, and that somewhere in the network a larger id exists. However, as P_i is the only node with $id = id_i$ whose tree is of size $> L$, it can capture its neighboring trees until either facing a larger id, or until covering the entire network. In the latter case, P_i knows for sure that it is the leader. (In case of meeting a larger id, the smaller id must give up.) Thus, if $S_i > L$ then P_i starts a broadcast and collate of $lead(id_i)$. If P_i receives $ack(lead(id_i))$ from all neighbors then it can conclude that it is the leader. If there is a unique leader (i.e., unique node with the maximum id), then this will eventually happen. At this point, P_i broadcasts a $halt$ message that causes all processors to halt.

The additional messages in Algorithm $ELECT2$ are: $< ack(id_j), s >$, $lead(id_j)$, $ack(lead(id_j))$ and $halt$. They are sent through the same links where messages id_j were sent in Algorithm $ELECT$. Thus, the message complexity is within a constant factor from that of Algorithm $ELECT$. Some of the messages are now of size $O(\log n)$, since the size of the network is delivered as well. These messages however are only sent along tree edges (when the count is not 0). Note that for each id we have in Part I of algorithm $ELECT2$ at most one forest (not necessarily spanning forest) in the network. The number of tree edges in each forest is at most $n-1$. By Claim 3, the expected number of distinct ids is $O(\log n)$, therefore, the expected number of messages of size $O(\log n)$ is at most $O(n \log n)$.

In Section 3 it was shown how to modify Algorithm $ELECT$ to get a reduced and worst case message complexity, when both lower and upper bounds on n are known to all nodes. The same modifications are employed in Algorithm $ELECT2$.

From the above we can conclude:

Theorem 2 *If n is known to be in the range $L < n \leq 2L$, then a leader can be elected with termination detection, with probability $> 1 - \epsilon$. The worst case message complexity is $O(mr \log^2 r)$, each of size $O(\log \log n + \log r)$ bits. In addition, there are $O(n)$ expected number of messages of size $O(\log n + \log r)$ bits.*

Note that with probability $< \epsilon$ no leader is elected. However, the fact that n is known within a factor of 2 is used to make sure that *at most* one leader can be elected. If we tolerate the existence, with small probability, of several leaders then termination detection can be achieved for the case that n is known within a factor of k, for any $k > 0$. Thus there are three possible failures: no leader is elected, no termination detection, or more than one leader is elected. In algorithm $ELECT2$, no leader is elected with probability $< \epsilon/3$. If at least one leader is elected then with probability $< \epsilon/3$ there will be no termination detection. A termination detection is always correct, i.e., a terminated processor will remain inactive and in this case there is at least one elected leader; however, with probability $< \epsilon/3$ the leader is not unique. The probability that *any* of the said failures occurs is at most ϵ. Thus, with probability $\geq 1 - \epsilon$ termination detection will occur and the elected leader is unique. We have:

Theorem 3 *If n is known to be in the range $L < n \leq kL$, then a leader can be elected, with worst case message complexity $O(mkr \log(rk) \log r)$, each message of size $O(\log \log n + \log(kr))$. In addition there are $O(n)$ expected number of messages of size $\log n$ bits. With probability $> 1 - \epsilon$ the leader is unique. If at least one leader is elected then there is termination detection.*

Proof: If $L < n \leq kL$, $k > 2$, then there might be several trees with the same id, all of size $> L$. However, there are at most $k-1$ such trees. Thus, for each value of id there

```
Procedure Initialize(i)                                          /* performed upon waking up */
call Choose(< t_i, s_i >);
t'_i := t_i − thresh; id_i :=< t'_i, s_i >;              /* thresh = log L − log(12 ln(4r)) */
max_i := id_i; leader :=true; captured :=false; S_i := 1;
unmark all links;
if t_i > t̄' then t_i := t̄';                  /* t̄' = log(2rL) − thresh = log(24r ln(4r)) */
if t_i > thresh then
    send id_i to all neighbors.

Upon receiving id at node i over link l:
if id_i not defined then call Initialize(i);
if id < max_i then ignore;
if id = max_i then send < ack(id), 0 > on l.                   /* link l is a non-tree edge */
if id > max_i then                                         /* P_i becomes a node of the tree */
        max_i := id; leader :=false; parent_i := l; S_i := 1;
        unmark all links (as 'not-acknowledged');
        send id to all neighbors.

Upon receiving < ack(id), k > at node i over link l:
if id < max_i then ignore;                                 /* the ack' is not relevant any more */
if id = max_i then:
S_i := S_i + k; mark l (as 'acknowledged');
if all neighbors are acknowledged then                   /* collated ack's from all neighbors */
        if leader =false then                                              /* i is not the root */
                send < ack(id), S_i > on link parent_i;
        else                   /* i is the root; COLLATE is finished; S_i is the size of the tree */
                if S_i > L then                                             /* largest tree with id */
                        unmark all links (as 'not-acknowledged');
                        captured := true;
                        send lead(id_i) to all neighbors;
                else leader := false;
```

Figure 4: $ELECT2$ - Part 1

are at most $k-1$ nodes that initiate the message $lead(id)$. Let each one of these nodes randomly select a $label$ from the range $[1..4k^2/\epsilon]$. The $lead$ message will now consist of the pair $< lead(id), label >$. All labels are distinct with probability $> 1-\epsilon/2$. Therefore, with probability $> 1-\epsilon$, a unique leader is elected. Termination is detected if at least one node considers itself a leader since any such node floods a halt message following which any node stops receiving messages of the algorithm.

The message complexity is, up to a factor of $(k^2 r)$, the same as in the respective algorithm

```
Upon receiving lead(id) at node i over link l:
if id < max_i then ignore;
if id = max_i then
        if captured =true then                              /* link l is a non-tree edge */
                send ack(lead(id)) on l;
        else
                captured := true; parent_i := l;
                unmark all links (as 'not-acknowledged');
                send lead(id) to all neighbors.
if id > max_i then                                 /* P_i becomes a node of the tree T' */
        max_i := id; leader :=false; captured :=true; parent_i := l;
        unmark all links (as 'not-acknowledged');
        send lead(id) to all neighbors.

Upon receiving ack(lead(id)) at node i over link l:
if id < max_i then ignore;                         /* the ack' is not relevant any more */
if id = max_i then:
mark l acknowledged;
if all neighbors are acknowledged then            /* collated ack's from all neighbors */
        if leader =false then                                      /* i is not the root */
                send ack(lead(id)) on link parent_i.
        else                                        /* i is the root; COLLATE is over */
                send halt to all neighbors.
                get into state 'halt'.

Upon receiving halt over link l:
                send halt to all neighbors but l;
                get into state 'halt' (i.e., ignore future messages).
```

Figure 5: $ELECT2$ - Part 2

without termination detection in Section 3. The size of each message is, $O(\log n + \log(kr))$. ∎

Remark on electing a unique leader with no error: An algorithm with termination detection for the case that $L < n \leq 2L$ was given above. We saw that a necessary condition for a leader to be elected is that a tree of size at least L is constructed. It is possible to elect a unique leader with probability 1 in this case. For this purpose we need to detect a failure in the election algorithm so that an election procedure can be started all over again. We denote each election procedure as a *phase*.

When a root in phase i learns that its tree size is less than L it detects a failure in nominating itself a leader, and starts a new phase $i+1$. Eventually, in some phase a unique maximum id will be selected and a leader will be elected. However, even if a root in phase i fails to elect itself a leader, another root in the same phase i at a distant location in the network could still succeed. For this reason a node that starts participation in a new phase, still participates in any lower number phase. Thus, for each new phase an independent copy of algorithm $ELECT2$ is run in the network.

Different leaders may be elected in different phases. The leader of the algorithm is elected in the lowest number phase, in which a unique maximum id was selected. The leader will be the node selecting this number. To enable this, we require that a node that is a root in phase i will not participate in a higher numbered phase before becoming inactive in phase i. Clearly, a leader elected in phase i will not enable any other leader to be elected in higher numbered phase. Once a node received a lead message of phase i it can stop the execution of any higher numbered phase. Still if in some phase a leader is elected its id is the largest id chosen in this phase. The expected message complexity of this algorithm is $O(m \log n)$. However, we believe that under a "realistic" scheduler this algorithm performs well. The problem of defining a realistic scheduler and analyzing the protocol under such a scheduler is not dealt with here and is left as an open question. Note that in (Schieber and Snir, 1989) an $O(m+n\log n)$ messages algorithm for electing a leader with probability 1 under the same conditions is presented.

5 Election in complete networks

Consider the special case where the network is a complete graph. Using the previous results we obtain message complexities of more than $O(m) = O(n^2)$. In this section we show that the leader election problem in complete anonymous networks is as easy as in anonymous networks with a linear number of links (note that the situation is similar in the non-anonymous networks (Korach *et al.*, 1984; Afek and Gafni, 1991). Specifically, we present an algorithm whose message complexity is $O(n \cdot r \log(rk) \log r)$, which for fixed $r, k > 0$ is $O(n)$. This should be contrasted with the $\Omega(n \log n)$ lower bound for deterministic leader election in complete networks (Korach *et al.*, 1984), which holds even in the synchronous case (Afek and Gafni, 1991).

The main idea is the following: given a network G we distributively select a sparse random subgraph G' that spans G and, with a very high probability, is connected. We apply the appropriate algorithm for leader election on G'. The elected leader may also be a leader of G. The complexities are now functions of n, m' and D', where m' is the number of links in G' and D' is the diameter of G'. Our main concern is, therefore, to distributively find G' with m' as small as possible.

A simple algorithm for finding such G' would be for each node in G to randomly select q out of its $n-1$ incident links. Let $l=(v,u)$ be an edge of G' if and only if l was selected by either v or u. Clearly, for each fixed q, $m' \leq qn = O(n)$. Fenner and Frieze (1982) showed the following theorem which guarantees that G' is connected with high probability.

Theorem 4 (Fenner and Frieze, 1982) *Let G' be an undirected graph in which each vertex randomly chooses q neighbors, then for each fixed $q \geq 3$ the probability that G' is connected tends to 1 as n tends to infinity.*

Recall that the time complexity in all our algorithms is $O(D)$. For a complete graph this means $O(1)$ time. When using G' instead of G in order to reduce the message complexity, the penalty is not too high: it can be shown (Alon, 1989) that for fixed q and ϵ, the diameter of G' is $O(\log n)$ with probability $\geq 1-\epsilon$.

6 Discussion

Reduced complexities for low diameter networks: If the network is synchronous then the largest *id* captures the entire network in $O(D)$ worst case time and $O(mD)$ worst case messages. Thus, by using synchronizer α of Awerbuch (1985a) we guarantee that the message complexity in Algorithm $ELECT$ is $O(mD)$.

Variants of the broadcast method: All the election algorithms given in the previous sections are based on two steps. In the first each node selects an *id*, and in the second some broadcast method is applied in order to let the leader's *id* reach every node in the network. We give here two variants for the broadcast method; these variants can be applied to all the above algorithms.

The following rule will reduce the message complexity for many network topologies (but not in the worst case). Before actually sending an *id* to all neighbors, the node receives each of their max_i. If any of its neighbors has $max_i > id$, then the node purges *id* without broadcasting it. It is equivalent to the broadcast we used only that a root is required to perform a local collate on the messages it sent to distance 1 before continuing with the process. The broadcast proceeds only if the messages to distance 1 did not meet any larger *id*. (This idea can be extended to distances larger than 1.)

A second variant is to replace the Breadth First Search type behavior of Algorithm $ELECT$ by a Depth First Search type algorithm, similarly to the linear-time-DFS Algorithm in (Awerbuch, 1985b). Intuitively, the number of nodes visited by an *id* should be smaller in a DFS-type broadcast, relatively to a BFS-type broadcast. In fact, it is easy to see that this number is maximal in DFS-type broadcast when the topology is a ring.

Extensions: Based on an earlier version of this paper (Matias and Afek, 1989) several extensions of our method were obtained. Itai (1990) implements our algorithm in a ring of probabilistic finite automata, incurring the same communication complexity. Lavallee and Lavault (1990) extend our algorithm from Section 2 to achieve some improvement in the message complexity (from $O(m \log n)$ to $O(m \log \log n)$) for the cost of increasing the time complexity. Our *id* selection procedure is also used in (Afek *et al.* 1990) to elect a leader in a self-stabilizing manner.

Acknowledgment: We thank Noga Alon, Baruch Schieber and Michael Merritt for helpful discussions. In particular, Noga Alon pointed out to us the results of Section 2.2 and the reference (Fenner and Frieze, 1982). We also thank the anonymous referees for several helpful comments.

References

[1] [Abrahamson, K., Adler, A., Gelbart, R., Higham, L., and Kirkpatrick, D. (1989)] The bit complexity of randomized leader election on a ring. *Siam J. on Computing*, 18(1):12–29.

[Abrahamson, K., Adler, A., Higham, L., and Kirkpatrick, D. (1986)] Probabilistic solitude verification on a ring. In *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing.*

[Afek, Y., Awerbuch, B., Plotkin, S., and Saks, M. (1987)] Local management of a global resource in a communication network. In *Proc. of the 28th IEEE Annual Symp. on Foundation of Computer Science*, pages 347–357.

[Afek, Y. and Gafni, E. (1991)] Time and message bounds for election in synchronous and asynchronous complete networks. *Siam J. on Computing*, 20(2):376–394. Also in PODC 85.

[Afek, Y., Kutten, S., and Yung, M. (1990)] Memory-efficient self stabilizing protocols for general networks. In *Proc. of the 4th International Workshop on Distributed Algorithms: Springer-Verlag LNCS 486*, pages 15–28.

[Alon, A. (1989)] Private Communication.

[Angluin, D. (1980)] Local and global properties in networks of processes. In *Proc. of the 12th Ann. ACM Symp. on Theory of Computing*, pages 82–93.

[Attiya, H., Snir, M., and Warmuth, K. M. (1988)] Computing on an anonymous ring. *J. of the ACM*, 35(4):845–875.

[Angluin, D. and Valiant, L. G. (1979)] Fast probabilistic algorithms for hamiltonian paths and matchings. *J. Comp. Syst. Sci.*, 18:155–193.

[Awerbuch, B. (1985)] Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823.

[Awerbuch, B. (1985)] A new distributed depth-first-search algorithm. *Information Processing Letters*, 20(3):147–150, April 1985.

[Awerbuch, B. (1987)] Linear time distributed algorithms for minimum spanning trees, leader election, counting and related problems. In *Proc. of the 19th Ann. ACM Symp. on Theory of Computing*, pages 230–240.

[Burns, J. E. (1980)] A formal model for message passing systems. Technical Report TR-91, Indiana Univ.

[Chang, E. J. (1979)] *Decentralized Algorithms in Distributed Systems*. PhD thesis, University of Toronto.

[Chernoff, H. (1952)] A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Statistics*, 23:493–507.

[Cole, R. and Vishkin, U. (1986)] Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proc. of the 16th Ann. ACM Symp. on Theory of Computing*, pages 206–219.

[Fenner, T.I. and Frieze, A.M. (1982)] On the connectivity of random m-orientable graphs and digraphs. *Combinatorica*, 2:347–359.

[Frederickson, G. and Santoro, N. (1986)] Breaking symmetry in synchronous networks. In *VLSI Algorithms and Architectures, AWOC*, pages 82–93, 1986. Lecture Notes in Computer Science, No 227, Spring Verlag.

[Gallager, R. G., Humblet, P. A., and Spira, P. M. (1983)] A distributed algorithm for minimum weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5:66–77, January 1983.

[Itai, A. and Rodeh, M. (1990)] Symmetry breaking in distributed networks. *Information and Computation*, 88:60–87. See also FOCS-81, 150-158.

[Itai, A. (1990)] On the computational power needed to elect a leader. In *Proc. of the 4th International Workshop on Distributed Algorithms: Springer-Verlag LNCS 486*, pages 29–40.

[Korach, E., Moran, S., and Zaks, S. (1984)] Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In *Proc. of the ACM Symp. on Principles of Distributed Computing*, pages 199–207.

[LAVALLEE, I. AND LAVAULT, C. (1990)] Spanning tree construction for nameless networkds. In *Proc. of the 4th International Workshop on Distributed Algorithms: Springer-Verlag LNCS 486*, pages 41–56.

[MATIAS, Y. AND AFEK, Y. (1989)] Simple and efficient election algorithms for anonymous networks. In *Proceedings of the 3rd International Workshop on Distributed Algorithms: Springer-Verlag LNCS 392*, pages 183–194.

[PACHL, J. (1985)] A lower bound for probabilistic distributed algorithms. Technical Report CS-85-25, University of Waterloo, Waterloo, Ontario, Canada.

[SEGALL, A. (1983)] Distributed network protocols. *IEEE Trans. on Information Theory*, IT-29(1).

[SCHIEBER, B. AND SNIR, M. (1989)] Calling names on nameless networks. In *Proc. of the ACM Symp. on Principles of Distributed Computing*, pages 319–328. To appear in Information and Computation.