Scalable Leader Election

Valerie King [*] Jared Saia [†] Vishal Sanwalani [†] Erik Vee [‡]

Abstract

In the leader election problem, there are n processors of which $(1-b)n$ are good. The problem is to design a distributed protocol to elect a good leader from the set of all processors. In this paper, we present a scalable leader election protocol. Our protocol is *scalable* in the sense that each good processor sends and processes a number of bits which is only polylogarithmic in n. (We assume no limit on the number of messages sent by bad processors.) For $b < 1/3$, our protocol elects a good leader with constant probability and ensures that a $1 - o(1)$ fraction of the good processors know this leader.

We assume a *point-to-point full information model*. This is similar to the full information model, but harder in the sense that in a given round, a bad processor may send different messages to different processors, rather than having to broadcast the same message to every processor.

To the best of our knowledge, we present the first leader election protocol that ensures that each good processor sends and processes a sublinear number of bits. Having reduced the problem of leader election to one of informing all good processors of a bit held by $1 - o(1)$ fraction of good processors, we conjecture that the solution to this problem is not possible within polylogarithmic message bounds.

Our techniques can be used to provide scalable solutions to Byzantine agreement and other problems.

1 Introduction

Leader election is a fundamental problem in distributed computing. We consider this problem in the setting where there are n processors, bn of which are *bad* (or corrupt), and $(1-b)n$ of which are *good*, for some fixed b. Our goal is to design an algorithm that ensures a good processor will be elected with constant probability, no matter which set of bn processors are bad. This problem was first formally described and addressed by Ben-Or and Linial [4, 5] about twenty years ago. Since then, many papers have been published giving leader election algorithms that successively improve on the number of rounds required and on the fraction b of bad processors that can be tolerated [18, 1, 2, 7, 6, 9, 15, 17, 12] (see also surveys by Ben-Or, Linial and Saks [3] and Linial [14]).

In this paper, we describe a leader election algorithm which is *scalable*, in the sense that each good processor sends and processes a number of bits which is only polylogarithmic in n. Our motivation for this problem is the emergence of large-scale networks (e.g. peer-to-peer) where n can be on the order of hundreds of thousands.

1.1 The model The standard model for communication in the leader election problem is the *full information model*[1] [4, 5]. In this model, all communication occurs by broadcast and is known publicly to all processors. Every processor has a unique identity known to everyone and the identity of the sender of any message is explicitly known by all processors. Communication occurs in *rounds*. In each round, every processor may communicate with all other processors. The bad processors are assumed to have received the messages of all the good processors before they broadcast their own messages. The processors are synchronized between rounds so that all messages in round i are assumed to be received before any messages in round $i+1$ are sent out. Each processor has access to private random bits that are not known to other processors or the adversary.

round-based model of communication

The adversary picks which processors will be bad before the algorithm begins and controls the actions of all bad processors so as to maximize the chance of getting a bad processor elected. Also the adversary is computationally unbounded which disallows the use of cryptographic assumptions.

The full information model rules out from the very start any possibility of designing an algorithm where each processor sends a sublinear number of bits. In

[*] Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, BC, Canada V8W 3P6; email: val@cs.uvic.ca. This research was supported by NSERC.

[†] Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386; email: {saia,vishal}@cs.unm.edu. This research was partially supported by NSF grant CCR-0313160 and Sandia University Research Program grant No. 191445.

[‡] IBM Almaden Research Center, 650 Harry Road, San Jose, Ca. 95120; email: vee@almaden.ibm.com

[1] This is sometimes also referred to as the *perfect information model.*

SODA '06, January 22-26, Miami, FL
©2006 SIAM ISBN 0-89871-605-5/06/01

particular, since all communication is by broadcast, every time a node communicates in this model, it sends out $\Omega(n)$ bits. To avoid this problem, we introduce the *point-to-point full information model.* In this model, all communication occurs between a single sender and a single receiver. The bad processors see all messages but the good processors only see messages that are sent directly to them. Everything else is the same as in the full information model. In particular, each processor has a unique identity that is known explicitly to anyone to whom it sends messages. Further, communication occurs in rounds and the bad processors see all messages before they need to send their own. This new model is strictly harder than the standard full information model in the following sense. In the standard model, in a single round, a bad processor is forced to send the same message to all processors (since communication is by broadcast). However, in the new model, a bad processor can send different messages to different processors.

Our goal is to limit the number of bits *sent* and *processed* by every good processor. We assume that a processor can choose to ignore (not process), without cost, messages received from any other processor during any round of the algorithm. We do not assume a limit on the number of messages sent by a bad processor.

1.2 Our Results

In this paper, we relax the requirement that *all* good processors know the leader at the end of the protocol to the requirement that a $1 - o(1)$ fraction of good processors know the leader. Our main result is stated as Theorem 1.1 below. To the best of our knowledge, this is the first result for the leader election problem where each processor sends and processes a sublinear number of bits.

THEOREM 1.1. *Suppose there are n processors and strictly less than a $1/3$ fraction of these processors are bad. Then there exists an algorithm that elects, with constant probability, a leader from the set of good processors and has the following properties.*

- *Exactly one good processor considers itself the leader.*
- *A $1-o(1)$ fraction of the good processors know this leader.*
- *Every good processor sends and processes only a polylogarithmic (in n) number of bits.*
- *The number of rounds required is polylogarithmic in n.*

Given that almost all processors know the leader, a natural followup step is to have each good processor sample the other processors and then determine the leader from the majority of the responses. With high probability, such sampling would work, except for one issue– the good processors which are sampled may be swamped with requests from bad processors and may not know which queries to respond to. It is easy to get around this problem if we restrict the number of messages sent by bad processors and assume private communication channels between all pairs of processors. However, we see no way to get around the problem without making additional assumptions and thus we propose the following:

CONJECTURE 1. *Suppose $1-b$ fraction of processors are good, for any constant $b < 1/2$, and $1 - o(1)$ fraction of good processors agree on a bit. In the point-to-point full information model which allows the bad processors to send an unlimited number of messages and limits each good processor to sending $o(n)$ bits, there is no protocol which will ensure that with constant probability* all *good processors learn that bit.*

We note that our scalable leader election algorithm directly leads to a scalable algorithm for the more general selection task problem [12] (which includes e.g. collective coin flipping). In addition, a modification to our scalable leader election algorithm leads to a scalable algorithm for the Byzantine agreement problem which we now sketch [2]. First we modify our leader election algorithm to elect a set, S, of $\Theta(\ln n)$ processors such that, with high probability, S contains at least a $2/3$ fraction of good processors. Next, the processors in S use a polynomial time Byzantine agreement algorithm, e.g., [10], to come to consensus on a bit. Finally, this bit is sent out to all but a $1 - o(1)$ fraction of the good processors using techniques similar to those described in Section 4. Full details of the modifications required are omitted from this extended abstract.

The algorithm we use in Theorem 1.1 constructs a structure we call a *communication tree*. The communication tree allows messages to be passed from the leader, sitting at the root, to a $1 - o(1)$ fraction of the processors. It also allows messages from these processors to be sent to the leader. Since the communication tree appears to have applications beyond leader election, we include a theorem describing its properties.

The tree consists of nodes, and each node is assigned a set of processors. For now, call a node *good* if the set of processors assigned to it has a majority of good processors. [We will use a somewhat stricter notion of good in section 6.] Say a node has a *good path* to the root

[2] We thank Jonathan Katz and Chiu-Yuen Koo for pointing out this modification.

node if the path from the node to the root [including the root and the node itself] consists entirely of good nodes.

Theorem 1.2. *Suppose there are n processors, and a $b < 1/3$ fraction of these processors are bad. Then there is an algorithm operating in a number of rounds that is polylogarithmic in n and in which each good processor sends and processes only a polylogarithmic (in n) number of bits, that creates a communication tree with the following properties.*

(1) The tree has height ℓ^*, where ℓ^* is the minimum integer such that $n/\ln^{\ell^*} n \leq \ln^{10} n$. (Note that $\ell^* = O(\ln n/\ln\ln n)$.) The tree is rooted at the single node at level ℓ^*. A non-root node x from level $\ell > 0$ has as its children $\ln n$ nodes from layer $\ell - 1$.

(2) Each leaf node of the tree is assigned a set of $\ln^5 n$ processors. Each processor is assigned to $O(\ln^4 n)$ leaf nodes. All processors know exactly which processors participate in which leaf nodes.

(3) Each internal node of the tree other than the root is assigned a set of $\ln^3 n$ processors. Each processor is assigned to at most $O(\ln^4 n)$ internal nodes per level. The root node is assigned a single processor.

(4) With probability $1 - o(1/n)$, at most a $3/\ln^2 n$ fraction of the nodes in the tree are bad, and all but a $3/\ln n$ fraction of the leaf nodes have a good path to the root node.

(5) With constant probability, the single processor at the root node is good.

(6) Suppose there is a good path from A to the root, and let B be the parent of A. Then all processors in A know all processors in B, and all processors in B know all processors in A.

1.3 Related Work The first results for leader election in the full information model are due to Ben-Or and Linial [4, 5]. They give a one round protocol which is robust against up to a $1/\ln n$ fraction of bad processors. Later results [13, 18, 1] improved the fraction of bad processors that could be tolerated, culminating with a protocol due to Alon and Naor [2] which was shown by Boppana and Narayanan [7, 6] to be robust to bn bad processors for all $b < 1/2 - \epsilon$ and positive ϵ. A result by Saks [18] shows that no protocol can be robust against $\lceil n/2 \rceil$ bad processors.

The protocol due to Alon and Naor has optimal resilience but requires a linear number of rounds. Several subsequent papers focused on reducing the number of rounds [9, 15, 19, 17]. These papers culminated in protocols which are resilient against $b < 1/2 - \epsilon$ bad processors and require only $\ln^* n$ rounds [16, 12]. A result by Russell, Saks and Zuckerman [16] shows that $\Omega(\ln^* n)$ rounds are necessary if in every round the correct processors each send one unbiased random bit. We emphasize again that all of these protocols require each processor to send and process $\Omega(n)$ messages.

Our scalable leader election algorithm makes use of several past results. First, we make use of the concept of a *committee* which is a subset of processors. The notion of a committee was introduced by Bracha [8] and used in e.g. [9, 15, 17]. Second, our algorithm uses an adaptation of Feige's leader election algorithm [12] as a subroutine, as described in Lemma 2.1. Finally, we make use of a Byzantine Agreement algorithm designed by Dolev et al. [10]; details of how this algorithm is used are described in Lemma 2.1.

The notion of *almost-everywhere* Byzantine agreement in which not all but almost all correct processors are required to come to agreement was introduced in a paper by Dwork, et. al. [11] on fault tolerance in bounded degree networks.

1.4 Overview Throughout the paper, we will use the phrase *with high probability* (or simply *w.h.p.*) to mean that an event happens with probability at least $1 - o(n^{-c})$ for some $c > 2$. For readability, we treat $\ln n$ as an integer.

We now give a high-level sketch of how our algorithm works. In its simplest form, we divide the processors into groups of polylogarithmic size; each processor is assigned to multiple groups. In parallel, each group then elects a small number of processors from within their group to move on. We then recursively repeat this step on the set of elected processors until the number of processors left is polylogarithmic. At this point, the remaining processors run one more election to determine the leader.

Although this approach is intuitively simple, there are several complications that must be addressed.

(1) Each group must be able to robustly hold an election.

(2) The groups must be determined in such a way that the election mechanism cannot be sabotaged by the bad processors.

(3) After each step, each elected processor must determine the identities of certain other elected processors, in order to hold the next election.

(4) Election results must be communicated to the processors.

(5) To ensure load balancing, a processor which wins too many elections in one round cannot be allowed to participate in too many groups in the next round.

We address (1) in section 2. Since each group is of polylogarithmic size, the processors are able to robustly simulate a broadcast using a polylogarithmic number of message bits. They are then able to use known protocols for leader election designed for the broadcast model.

Item (2) is addressed in section 3. In order to make the group assignments, we use a layered network with extractor-like properties. Every processor is assigned to a specific set of nodes on layer 0 of the network. In order to assign processors to a node A on layer $\ell > 0$, the set of processors assigned to nodes on layer $\ell - 1$ that are connected to A hold an election. In other words, the topology of the network determines how the processors are assigned to groups. By choosing the network to have certain desired properties, we can ensure that the election mechanism is robust against malicious adversaries.

To accomplish item (3), we use *monitoring sets*, which we describe more fully in section 4. Each node A of the layered network is assigned a set of nodes from layer 0, which we denote $m(A)$. (Abusing notation, we also let $m(A)$ denote the set of processors assigned to nodes of $m(A)$.) The job of the processors from $m(A)$ is simply to know which processors are assigned to node A. Since the processors of $m(A)$ are fixed in advance and known to all processors, any processor that needs to know which processors are assigned to A can simply ask the processors from $m(A)$. (In fact, the querying processor only needs to randomly select a polylogarithmic subset of processors from $m(A)$ in order to learn the identities of the processors in A with high probability. This random sampling will be used to ensure load balancing.) Since the number of processors that need to know the identities of processors in node A is polylogarithmic, the processors of $m(A)$ will not need to send too many messages.

We use a *communication tree* to inform the monitoring sets which processors won each of their respective elections, as well as to inform all processors who won the final leader election, thus addressing item (4). The important properties of the communication tree are given in Theorem 1.2, and the communication tree itself is described in section 4.

Item (5) is addressed by having such processors refrain from further participation. We show there are not many such processors in section 5. In section 5 we described the Leader Election Algorithm. Our proof of Theorem 1.1 and 1.2 is presented in section 6.

2 Subcommittee elections

We now describe how small groups of processors elect leaders and subcommittees. The idea is to use previously known protocols designed in the broadcast model, by simulating broadcast when necessary.

We first adapt an algorithm by Feige [12] to the point-to-point full information model to get what we will call the Heavyweight-Leader-Election protocol. This algorithm elects a good leader with constant probability from among a set of n processors, of which a fraction greater than $2/3$ are good. Feige's result shows that this can be done in $\ln^* n$ expected rounds, with each processor broadcasting at most once per round. To adapt Feige's result to the point-to-point full information model, we simulate broadcast. Each time it is necessary for processor p to broadcast in the original protocol, we replace it with p sending a message to all other processors followed by a call to a Byzantine Agreement algorithm such as [10], so that all processors agree on the message p sent. (The algorithm presented in [10] requires $O(k)$ rounds and $O(k^3)$ bits when there are k processors, and less than $k/3$ are bad. We will use $k = O(\log^{10} n)$ and $k = O(\log^8 n)$.) This results in the following:

Lemma 2.1. *[12] In the point-to-point full information model, there is a protocol we call* Heavyweight-Leader-Election *with the following properties. On a set k processors, with $(2/3 + \epsilon)k$ good processors, it returns a good leader with probability at least a positive constant and requires $O(k^4 \ln^* k)$ bits and $O(k^2 \ln^* k)$ rounds.*

Next, we describe a method for electing a subcommittee of processors, with desired properties from a committee of processors using what we call Elect-Subcommittee. This protocol is also a simple adaptation of a subroutine in [12].

Elect-Subcommittee: *Input is processors $p_1, \ldots, p_k$ with $k = \Omega(\ln^8 n)$.*

```
For $i = 1$ to $k$,
    Processor $p_i$ randomly selects one of $k/(c_1 \ln^3 n)$
    "bins" and tells the other processors in its
    committee which bin it has selected.
    The other processors in the committee run
    Byzantine Agreement to come to a consensus on
    which bin $p_i$ has selected.
Let $B$ be the bin with the least number of processors
in it, and let $S_B$ be the set of processors in that bin.
Arbitrarily add enough processors to $S_B$ to ensure
$|S_B| = c_1 \ln^3 n$.
Return $S_B$ as the elected subcommittee.
```

Lemma 2.2. *Let S be a committee of $\Omega(\ln^8 n)$ processors, where the fraction, f_S, of good processors is greater than $2/3$. Then there exists some constant c_1, such that w.h.p., the subcommittee election protocol elects a subset Z of S such that $|Z| = c_1 \ln^3 n$ and the fraction of good processors in Z is greater than $(1 - 1/\ln n)f_S$. Further, this algorithm uses a polylogarithmic number of bits and polylogarithmic number of rounds.*

Proof. The proof follows from a straightforward application of Chernoff and union bounds and is omitted in this extended abstract.

3 The layered network

We now describe the layered network we use to assign processors to groups that is robust against malicious adversaries.

We first present a result similar to that used in [9]. Let X be a set of processors. For a collection $\mathcal{F}$ of subsets of X, a parameter δ, and a subset X' of X, let $\mathcal{F}(X', \delta)$ be the subcollection of all $F' \in \mathcal{F}$ for which

$$\frac{|F' \bigcap X'|}{|F'|} > \frac{|X'|}{|X|} + \delta.$$

Let $\Gamma(r)$ denote the neighbors of node r in a graph. The following lemma is easily shown using the probabilistic method. We omit its proof here.

Lemma 3.1. *Let ℓ^* be the smallest integer such that $n/\ln^{\ell^*} n \leq \ln^{10} n$. There is a family of bipartite graphs $G(L_i, R_i), i = 0, 1, \ldots, \ell^*$, such that $|L_i| = n/\ln^i n$, $|R_i| = n/\ln^{i+1} n$, and*

1. *Each node in R_i has degree $\ln^5 n$.*
2. *Each node in L_i has degree $O(\ln^4 n)$.*
3. *For any subset L_i' of L_i, let $\mathcal{F} = \{\Gamma(r) \mid r \in R_i\}$. Then $|\mathcal{F}(L_i', 1/\ln n)| < |L_i|/\ln^3 n$.*
4. *Additionally, label the vertices in R_i by $0, 1, \ldots, |R_i|$ and the vertices in L_i by $0, 1, \ldots, |L_i|$. Then the vertices in L_i labelled $k \ln n, k \ln n + 1, \ldots, (k + 1) \ln n - 1$ are incident to the vertex labelled k in R_i.*

We are now ready to describe the layered network. Let ℓ^* be the minimum integer ℓ such that $n/\ln^\ell n \leq \ln^{10} n$; note that $\ell^* = O(\ln n/\ln \ln n)$. The topmost layer ℓ^* has a single node which is adjacent to every node in layer $\ell^* - 1$. For the remaining layers $\ell = 0, 1, ..., \ell^* - 1$, there are $n/\ln^{\ell+1} n$ nodes. There is an edge between the ith node in layer ℓ and the jth node in layer $\ell + 1$ iff there is an edge between the ith node in $L_{\ell+1}$ and the jth node in $R_{\ell+1}$ from Lemma 3.1.

Each node in the layered network will contain a set of processors known as a *committee.* All nodes, except for the one on the top layer and those in layer 0, will contain $c_1 \ln^3 n$ processors. Initially, we assign the n processors to nodes on layer 0 using the bipartite graph $G(L_0, R_0)$ described in Lemma 3.1. The ith processor is a member of the committee contained in the jth node of layer 0 iff there is an edge in G between the i node of L_0 and the j node of R_0. Notice that every node on layer 0 has $\ln^5 n$ processors in it.

Nodes on higher layers have committees assigned to them during the course of the algorithm. Let A be a node on layer $\ell > 0$, let $B_1, \ldots, B_s$ be the nodes adjacent to A on layer $\ell - 1$ in the network, and suppose that we have already assigned committees to nodes on layers lower than ℓ. If $\ell < \ell^*$, we assign a committee to A by running Elect-Subcommittee on the processors assigned to $B_1, \ldots, B_s$, and assigning the winning subcommittee to A. (Note that we can run each of these elections in parallel.) If A is at layer ℓ^*, we assign a committee (which is a single processor) by running Heavyweight-Leader-Election on the processors in $B_1, \ldots, B_s$ and assigning the winner to A.

4 Communication

As processors move up the layered network, the processors assigned to a given node are not generally known to the other processors. This information is reliably provided by *monitoring sets*, each of which is assigned to monitor the election for one node, along with a communication tree to communicate the election results to the monitoring sets.

During the course of the protocol, we will sometimes speak of a node sending a message to a processor, or a node sending a message to another node. In order for node A to send a message to processor p, every processor in A sends the message to p. Processor p considers the message it received most often to be the message that A sent. Note that so long as the majority of processors in A are good and agree on a correct message, the message p receives is correct. In order for node A to send a message to node B, we simply have node A send the message to every processor in B. Again, so long as the majority of processors in A are good, every good processor in B knows the correct message. Notice that A must know the processors in B in order to carry out this protocol. More subtly, each processor receiving a message from node A must know the processors in A in order to take the majority of the correct set of processors. We will ensure throughout that this is the case, w.h.p.

The communication tree. We construct a rooted tree T whose node set is the set of nodes from the layered network. The tree is rooted at the single node at level ℓ^*. The children of this node consist of all nodes from layer $\ell^* - 1$. In general, node i from layer $\ell > 0$ has as its children the nodes $i \cdot \ln n, i \cdot \ln n + 1, \ldots, (i+1) \cdot \ln n - 1$ from layer $\ell - 1$. For a node A, we denote the subtree of T rooted at A by $T(A)$.

Although the communication tree and the network can be treated separately, we find it convenient to embed the tree in the network. Recalling item 4 from Lemma 3.1, we see that if node A on layer $\ell > 0$ is adjacent to the nodes $B_1, \ldots, B_s$ on layer $\ell - 1$, then the children of A in the communication tree consist of some subset of the nodes $B_1, \ldots, B_s$. The advantage of this becomes apparent in the following algorithm, used to send a message from A to the leaf nodes of $T(A)$. Since the processors are assigned to A based on an election using the processors from $B_1, \ldots, B_s$, we see that the processors of A already know the processors of A's children in the communication tree (under reasonable assumptions about how many processors have been corrupted; we formalize these notions in Section 6).

To avoid confusion, we use the term level when we are discussing the tree T, and the term layer when we are discussing the layered network.

```
SEND-MESSAGE-DOWN-TREE: Input is node $A$ with
message $m$.
If node $A$ is not on level 0, then
   Node $A$ sends message $m$ to each of its
   children $C_1, \ldots, C_t$ in the communication tree $T$.
   Call SEND-MESSAGE-DOWN-TREE on each $C_i$
   with message $m$.
```

Monitoring sets. For every node A, we define a monitoring set, denoted $m(A)$, that consists of nodes from layer 0. This set of nodes is determined before the start of the algorithm. Specifically, for node A on layer 0, $m(A)$ consists of the node A. For node A on layer $\ell > 0$, let $B_1, \ldots, B_s$ be the nodes adjacent to A on layer $\ell - 1$. The set $m(A)$ consists of the leaf nodes of $T(B_1), \ldots, T(B_s)$. Notice that the monitoring set for the single node on level ℓ^* consists of every layer-0 node. Since these sets are fixed, every processor knows the nodes in $m(A)$ throughout the protocol. Further, since $m(A)$ consists of layer-0 nodes, the processors in nodes of $m(A)$ are known by everyone throughout the algorithm. Thus, if processor p needs to know the identity of the processors in node A, it may communicate with $m(A)$ in order to learn that information.

One interesting aspect of this problem is that straightforward polling can be defeated by flooding. That is, suppose node A wants to know the identity of the processors in node B. Suppose A randomly samples a subset of $m(B)$ to get the required information. (In general, $m(B)$ contains too many nodes for A to talk to everyone in $m(B)$.) If the processors of A are unknown to $m(B)$, then every bad processor can request the same information. The good processors, having a limit on the number of bits they may send, do not know which requests to ignore; hence, the protocol would fail.

Communication with monitoring sets. Throughout the algorithm, nodes on the same layer will frequently need to know the processors assigned to each other. The following protocol allows a node A to determine the processors in node B on the same layer. It assumes that the election algorithm expects nodes A and B to communicate with each other. (Otherwise, processors in $m(B)$ could get flooded with requests from bad processors.) Although it will not be necessary for us, a simple extension also allows nodes on different layers to determine the processors assigned to each other.

When A needs to learn the processors in B, the nodes in $m(B)$ tell corresponding nodes in $m(A)$ who the processors in B are. Node A then queries the nodes of $m(A)$. Note that both the processors from nodes in $m(A)$ and the processors from nodes in $m(B)$ know each other. Since every processor from a node in $m(A)$ has a belief about which $O(\ln^3 n)$ processors are in A, they will answer at most $O(\ln^3 n)$ queries, from these processors.

```
LEARN-PROCESSORS: Input is nodes $A, B$, both from
the same layer.
1  For all $i = 1, 2, \ldots m(A)$,
       The $i$th node of $m(B)$ tells the $i$th node of
       $m(A)$ the identities of the processors in node $B$.
2  Every processor $p$ in $A$ randomly selects a set of
   $\ln^2 n$ nodes in $m(A)$ to poll.
       Each polled node sends $p$ the identities of the
       processors in nodes $B$ (according to the
       messages it received in step 1). Processor $p$
       determines the processors in node $B$ by the
       majority of messages sent by the nodes it
       polled.
```

5 The Leader Election Algorithm

We are now ready to present the leader election protocol.

```
LEADER-ELECTION
For $\ell = 1$ to $\ell^*$:
  For each node $A$ in layer $\ell$, (Let $B_1, \ldots, B_s$ be
  nodes adjacent to $A$ in layer $\ell - 1$ of the network.)
    Call LEARN-PROCESSORS on nodes $B_i, B_j$ for
    all $i, j \in [s]$.
```

```
4       If $\ell < \ell^*$, run ELECT-SUBCOMMITTEE on the
        processors in nodes $B_1, \dots, B_s$. Assign winning
        processors to node $A$.
5       Else, run HEAVYWEIGHT-LEADER-ELECTION
        on the processors in nodes $B_1, \dots, B_s$. Assign
        the single winning processor to node $A$.
6       Call SEND-MESSAGE-DOWN-TREE on each $B_i$
        to communicate the identities of the processor(s)
        in $A$ to the leaves of $T(B_1), \dots, T(B_s) = m(A)$.
7     If processor $p$ is elected to more than 8
      nodes on layer $\ell$, it becomes silent, refusing
      to participate in any more elections.
8   Every processor is assigned to more than one
    layer-0 node, hence receives multiple messages on
    who wins the election. Each processor takes the
    message it receives most often to be the true
    message.
9   Every processor randomly polls $\ln^2 n$ processors to
    determine the leader. A processor only responds to
    the processor it believes to be the leader, ignoring
    all other queries. If a majority of a processor's
    queries are answered, it considers itself the leader.
```

Step 7 of LEADER-ELECTION is simply to ensure the load-balancing condition of the main theorem. The fact that good processors drop out of elections throughout the algorithm seems like a potential problem. However, the following lemma shows that the effect of silent processors is not too great. We will use it later to show that very few elections are spoiled because of these silent processors.

LEMMA 5.1. *W.h.p. the fraction of nodes on layer ℓ that contain more than a $4/\ln n$ fraction of good processors that have become silent is at most a $4/\ln n$ fraction of the total nodes on layer ℓ.*

Proof. Let X be the number of processors on layer ℓ (counting multiplicity). The number of processors elected to layer $\ell+1$, counting multiplicity, is precisely $X/\ln n$. We show that w.h.p., the number of processors elected more than 8 times to layer $\ell+1$, counting multiplicity, is at most $16X/\ln^3 n$. Our claim will thus follow.

By construction, the number of nodes on layer $\ell+1$ is $X/\ln^4 n$, where the election for each employs $\ln^s n$ bins, with $s = 7$ for $\ell = 0$ and $s = 5$ otherwise. Being overly generous, suppose that for each election the adversary is able to choose which bin is elected. For a fixed sequence of $X/\ln^4 n$ bin choices (one for each of the $X/\ln^4 n$ elections held on layer $\ell+1$), let p_k be the probability that a given (good) processor is elected k or more times. Then for a fixed bin sequence– since each good processor independently and randomly chooses which bin to select– the probability that at least αX processors are elected k or more times is bounded by

$$\binom{X}{\alpha X} p_k^{\alpha X} \le \left(\frac{e p_k}{\alpha}\right)^{\alpha X}$$

Hence, by a union bound, the probability that any bin sequence causes at least αX processors to be elected k or more times is

$$\le \quad \left(\frac{e p_k}{\alpha}\right)^{\alpha X} \cdot (\ln^s n)^{X/\ln^4 n}.$$

Each node on layer ℓ participates in at most $2\ln^4 n$ elections. Since processors that are assigned to more than 8 nodes become silent, no processor on layer ℓ participates in more than $16\ln^4 n$ elections. So we may upper bound p_k by

$$\binom{16\ln^4 n}{k} \cdot \left(\frac{1}{\ln^s n}\right)^k \le \left(\frac{16e\ln^4 n}{k\ln^s n}\right)^k \le \left(\frac{16e}{k\ln n}\right)^k.$$

Notice that for $k > 8$ and n sufficiently large that $(16e/(k\ln n))^{k/2} \le 16/(k\ln^4 n)$. Hence, using $\alpha = 16/(k\ln^4 n)$, we see that for $k > 8$ and n sufficiently large that $e/\alpha \le p_k^{-1/2}$. From above, the probability that any bin sequence causes at least $16X/(k\ln^4 n)$ processors to be elected k or more times is at most

$$\begin{aligned}\left(\frac{e p_k}{\alpha}\right)^{\alpha X} \cdot (\ln^s n)^{X/\ln^4 n} &\le p_k^{\alpha X/2} \cdot (\ln^7 n)^{X/\ln^4 n} \\ \le \quad \left(\frac{16e}{k\ln n}\right)^{8X/\ln^4 n} &\cdot (\ln n)^{7X/\ln^4 n} = o(n^{-m}).\end{aligned}$$

Hence, w.h.p., the number of processors elected 9 or more times, counting multiplicities, is at most

$$16X/(\ln^4 n) + \sum_{k\ge 10} 16X/(k\ln^4 n) \le 16X/\ln^3 n$$

from which the lemma follows.

6 Proofs of Theorem 1.1 and 1.2

We focus on the proof of Theorem 1.2, since Theorem 1.1 follows as a consequence. Lemma 6.3 below will guarantee that the number of messages sent and the number of rounds taken by the algorithm is polylogarithmic in n. Further, notice that items (1) and (2) follow by construction, and item (6) follows by the way in which the algorithm executes. Item (3) also follows trivially, assuming the algorithm executes as described. (That is, so long as the adversarial processors cannot stop the algorithm from functioning, each internal node will be assigned $O(\ln^3 n)$ processors, with no processor assigned to more than $O(\ln^4 n)$ nodes per layer.) Lemma 6.1 below will show that, assuming the nodes of the network are not too corrupted, the algorithm works correctly. Lemma 6.2 shows that, in fact, the nodes of the network do not become too corrupted, w.h.p. Together, these show that the algorithm works as expected. Further, Lemma 6.2 proves item (4). Finally, item (5) of Theorem 1.2 will follow as a consequence of Lemma 2.1, Lemma 6.2, and the fact that the algorithm works as expected.

We recursively define the notion of good/bad nodes and good/bad leaves. A node A on layer 0 is a *bad node* if the fraction of bad processors in A is greater than $b + 12/\ln n$; a node is *good* if it is not bad. The node A (which is the unique leaf node of $T(A)$) is a *bad leaf for* $T(A)$ iff A is bad; otherwise, it is a *good leaf for* $T(A)$.

Now, consider a node A on layer $\ell > 0$, and let $B_1, \ldots, B_s$ be the nodes adjacent to A on layer $\ell - 1$. We say A is a *bad node* iff the fraction of processors in A that are bad is at least $b + 12(\ell + 1)/\ln n$, or if the fraction of leaf nodes from the trees $T(B_1), \ldots, T(B_s)$ that are bad leaves for their respective trees is greater than $3(\ell + 2)/\ln n$. If node C is a leaf of $T(A)$, then C is a *bad leaf for* $T(A)$ iff there is a bad node anywhere along the path from C to A in the tree (including A and C itself). For convenience, we will sometimes call C a *bad leaf for layer* ℓ in this case. If C is a leaf of $T(A)$ and it is not a bad leaf for $T(A)$, then C is a *good leaf for* $T(A)$. Likewise, C is a *good leaf for layer* ℓ in this case.

Finally, we call a node *too quiet* if at least a $4/\ln n$ fraction of its processors have become silent.

Lemma 6.1. *W.h.p., for all $\ell < \ell^*$, we have*

1. *Every call to* Learn-Processors *that is run on two good nodes A, A' on layer ℓ executes correctly.*
2. *Every call to* Send-Message-Down-Tree *that is run on a good node A will send the correct message to every good leaf node of $T(A)$.*
3. *Every call to* Elect-Subcommittee *that is run on processors from layer-ℓ nodes $B_1, \ldots, B_s$ elects a subcommittee with at most a $b + 12(\ell + 2)/\ln n$ fraction of bad processors, so long as at most a $7/\ln n$ fraction of the B_i's are bad or too quiet.*

Proof. We proceed by induction. The base case, $\ell = 0$, follows immediately for items 1 and 2. For item 3, suppose that at most a $7/\ln n$ fraction of the B_i's are bad or too quiet. So at most a $7/\ln n + 4/\ln n + b + 12/\ln n$ fraction of the processors in $B_1, \ldots, B_s$ are bad or silent. That is, at least a $(1 - b - 23/\ln n)$ fraction of the processors are good. By Lemma 2.2, the number of good processors assigned to A is at least a $(1-b-23/\ln n)(1-1/\ln n) > 1-b-24/\ln n$ fraction, as we wanted. Now assume the lemma holds for some $\ell \geq 0$, and consider $\ell + 1$.

We will first prove item 1 of the lemma. Let A, A' be good nodes on layer $\ell+1$, let $B_1, \ldots, B_s$ [respectively, $B'_1, \ldots, B'_s$] be the nodes adjacent to A [respectively, A'] on layer ℓ of the network. For each $i = 1, 2, \ldots, |m(A)|$, let D_i be the ith node in the monitoring set $m(A)$, and let D'_i be the ith node in the monitoring set $m(A')$. Since A and A' are both good nodes, the fraction of the D_i [respectively D'_i] that are bad leaf nodes for layer ℓ is at most $3(\ell + 2)/\ln n$. Hence, the fraction of i's such that either D_i or D'_i is a bad leaf node is at most $6(\ell + 2)/\ln n$.

Now, suppose D_i is a good leaf node for layer ℓ, and choose j so that D_i is a leaf node of $T(B_j)$. By induction (from item 2), when Send-Message-Down-Tree is called on B_j in step 6 of the Leader-Election protocol, D_i correctly learns the identity of the processors in A, w.h.p. Likewise, if D'_i is a good leaf node for layer ℓ, then it correctly learns the identity of the processors in A', w.h.p. Each processor p in A randomly selects $\ln^2 n$ nodes from $m(A)$. With probability at least $1 - 6(\ell + 2)/\ln n$, both the selected node in $m(A)$ and its corresponding node in $m(A')$ are good and know the identities of A and A', respectively. Hence, the expected number of messages that p receives that correctly name the processors in A' is at least $1 - 6(\ell + 2)/\ln n$. By a standard Chernoff bound, every processor in A learns the processors in A' w.h.p. Since Learn-Processors is called at most a polynomial number of times, it correctly works every time it is called on two good nodes w.h.p.

For item 2, let D be a good leaf node of $T(A)$, and let C be the child of A in the communication tree that lies on the path from D to A. By definition, C is good. Further, by the construction of the network, C participated in the election of the committee for A. So A knows the processors in C and will correctly pass the message to C. By induction, Send-Message-Down-Tree correctly sends the message from C down to D, w.h.p.

For item 3, let $B_1, \ldots, B_s$ be nodes on layer $\ell+1$. By induction (item 1), each good B_i knows the processors in each good B_j. We note that it is unimportant whether the processors in the good nodes from B_i agree on which processors map to the bad nodes from B_i. To see this, note that we are assuming that a single adversary controls all of the bad processors and that a bad processor can send different messages to different good processors. Thus a bad node can do no more mischief in the case where the good processors do not have the same view of which processor maps to it than in the case where the good processors all have the same view.

Now, suppose that at most a $7/\ln n$ fraction of the B_i are bad or too quiet. Hence, at most a $7/\ln n + 4/\ln n + b + 12(\ell+2)\ln n$ fraction of the processors from $B_1, \ldots, B_s$ are bad or silent. So, by Lemma 2.2, Elect-Subcommittee will elect a subcommittee with at most a $b + 12(\ell+3)/\ln n$ fraction of bad processors.

Lemma 6.2. *W.h.p., for all $\ell < \ell^*$, we have*

1. *The fraction of bad nodes on layer ℓ is at most $3/\ln^2 n$*
2. *The fraction of bad leaves for layer ℓ is at most $3(\ell+1)/\ln^2 n$.*

Proof. We prove the lemma by induction. The base case, $\ell = 0$, follows from Lemma 3.1. Assume the lemma holds for some $\ell \geq 0$ and consider $\ell + 1$.

Let A be a node from layer $\ell+1$, and let $B_1, \ldots, B_s$ be the nodes adjacent to A in the network on layer ℓ. Suppose that (1) the fraction of nodes $B_1, \ldots, B_s$ that are bad is at most $3/\ln^2 n + 1/\ln n$, (2) the fraction of nodes $B_1, \ldots, B_s$ that are too quiet is at most $5/\ln n$, and (3) the fraction of bad leaves from $T(B_1), \ldots, T(B_s)$ is at most $1/\ln n + 3(\ell+1)/\ln n + 1/\ln n$. By Lemma 6.1(item 3) and conditions (1) and (2), the fraction of bad processors assigned to A is at most $b + 12(\ell+2)/\ln n$ w.h.p. Further, the fraction of bad leaves for $T(B_1), \ldots, T(B_s)$ is at most $3(\ell+2)/\ln n$ by condition (3). That is, A is a good node, w.h.p.

We now calculate the fraction of such A from layer $\ell+1$ that violate either condition (1),(2), or (3). This upper bounds the fraction of bad nodes on layer $\ell+1$, w.h.p.

By induction, the number of bad nodes on layer ℓ is at most $3/\ln^2 n$. Hence, by Lemma 3.1, the fraction of nodes A from layer $\ell+1$ that have more than a $3/\ln^2 n + 1/\ln n$ fraction of $B_1, \ldots, B_s$ that are bad is at most $1/\ln^2 n$. That is, the number of nodes on layer $\ell+1$ that violate (1) is at most $1/\ln^2 n$.

Noting Lemma 5.1, we see that the fraction of nodes on layer ℓ that are too quiet is at most $4/\ln n$. We again invoke Lemma 3.1 to see that the fraction of nodes A that have more than a $4/\ln n + 1/\ln n$ fraction of their $B_1, \ldots, B_s$ that are too quiet is at most $1/\ln^2 n$. That is, the number of nodes on layer $\ell+1$ that violate (2) is at most $1/\ln^2 n$.

Finally, let $C_1, \ldots, C_r$ be the nodes on level ℓ. By induction, the total fraction of leaf nodes from among $T(C_1), \ldots, T(C_r)$ that are bad is at most $3(\ell+1)/\ln^2 n$. Hence, the number of i for which $T(C_i)$ has more than a $3(\ell+1)/\ln n$ fraction of bad leaf nodes is at most $1/\ln n$. Again, Lemma 3.1 guarantees that at most a $1/\ln^2 n$ fraction of the nodes on layer $\ell+1$ violate (3). Hence, the total fraction of bad nodes on layer $\ell+1$ is at most $3/\ln^2 n$.

To prove the second item, simply note that the fraction of bad leaves for layer $\ell+1$ is at most the fraction of bad leaves for layer ℓ plus the fraction of bad nodes on layer $\ell+1$. This is at most $3(\ell+1)/\ln^2 n + 3/\ln^2 n = 3(\ell+2)/\ln^2 n$, as we wanted.

To finish the correctness proof, consider the last iteration of the algorithm. From Lemma 6.2, the fraction of bad nodes on layer $\ell^* - 1$ is at most $3/\ln^2 n$. Hence, the fraction of bad processors on layer $\ell^* - 1$ is at most $3/\ln^2 n + b + 12/\ln n < 1/3 - \epsilon$ for some ϵ. Further, Lemma 6.1 guarantees that each good node on layer $\ell^* - 1$ correctly learns the processors in all of the other good nodes. So Heavyweight-Leader-Election elects a good leader with probability at least a positive constant, by Lemma 2.1. This proves item (5) of Theorem 1.2, thus completing the proof for Theorem 1.2.

In order to finish the proof for Theorem 1.1, we need to ensure that a $1 - o(1)$ fraction of the good processors know the results of the election. The results of the election are transmitted via calls to Send-Message-Down-Tree. By Lemma 6.1, all of the good leaves correctly receive the message. Since the fraction of bad leaves for level $\ell^* - 1$ is at most $3\ell^*/\ln^2 n < 3/\ln n$, the fraction of processors which are assigned to leaves so that more than a 1/3 fraction of the processor's leaves are bad is at most $9/\ln n$ fraction of the total nodes. Since the processors take the majority message, at least $1 - O(1/\ln n)$ processors correctly know the elected leader. Thus after the final polling step (step 9 of the algorithm), by the Chernoff bound, w.h.p. exactly one processor considers itself to be the leader.

Lemma 6.3. *The number of messages sent and processed by each good processor and the size of each message is polylogarithmic in n.*

Proof. By Lemma 3.1, each processor participates in at most $O(\ln n/\ln\ln n)$ layers. In each layer, each processor is involved in $O(\ln^4 n)$ elections, since a processor ceases to participate if it wins more than 8 elections on a layer. Every time an election is run, each participating processor sends polylogarithmic messages, by Lemma 2.2.

Also, each processor is in $O(\ln^5 n)$ monitoring set for each layer and communicates with $O(\ln^3 n)$ other processors each time monitoring sets communicate; monitoring sets communicate for each pair of siblings in the network. In addition, there are a number of messages passed down the communications tree. If A is a node at layer ℓ, and processor p is in a node of $m(A)$, then when Send-Message-Down-Tree is called on A, p must handle the message at most ℓ times. So the number of messages handled by each processor over the course of the entire algorithm is still polylogarithmic, and each message communicated in the tree T reports the election results of a single node, which requires $O(\ln^4)$ bits.

Finally, we note that the protocol involving random sampling can be used by the adversary to "flood" all the processors in the monitoring set that may be potentially sampled with requests by the processors doing the sampling, if the adversary has control of these processors (or their monitoring sets). However, this increases the load for each processor by only a polylogarithmic number of messages per processor, since only a polylogarithmic number of processors are identified as potential samplers.

7 Conclusion

In this paper, we have presented a scalable algorithm for the leader election problem. The algorithm is scalable in the sense that each good processor sends and processes a number of bits which is only polylogarithmic in n. For $b < 1/3$, our protocol elects a good leader with constant probability and ensures that a $1 - o(1)$ fraction of the good processors know this leader. To the best of our knowledge, this is the first leader election protocol which ensures that each good processor sends and processes a sublinear number of bits.

Several open problems remain, including the following. First, can we prove Conjecture 1 or some weaker version of this conjecture? Second, the number of rounds required by our algorithm and the number of bits sent and processed by good processsors are all polylogarithmic in n but the exponents of these logarithms are relatively large. Can we reduce these exponents? Can we reduce the fraction of good processors needed to do leader election below 2/3, as in [12]? Finally, can a simplified version of our algorithm or heuristics based on it be used in a practical large-scale network?

References

[1] M. Ajtai and N. Linial. The influence of large coalitions. Technical Report 7133(67380), IBM, 1989.

[2] N. Alon and M. Naor. Coin-flipping games immune against linear-sized coalitions. In *Proceedings of the IEEE Foundations of Computer Science(FOCS)*, 1990.

[3] Ben-Or, N. Linial, and M. Saks. Collective coin flipping and other models of imperfect randomoness. In *Colloq. Math Soc. Janos Bolyai No., 52 Combinatorics*, 1987.

[4] Michael Ben-Or and Nathan Linial. Collective coin flipping and other models of imperfect randomness. In *Proceedings of the IEEE Foundations of Computer Science(FOCS)*, 1985.

[5] Michael Ben-Or and Nathan Linial. Collective coin flipping. *Advances in Computing Research*, pages 91–115, 1989. JAI Press; Silvio Micali, editor.

[6] R. Boppana and B. Narayanan. Collective coin flipping and leader election with optimal immunity. manuscript.

[7] R. Boppana and B. Narayanan. The biased coin problem. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, 1993.

[8] Gabriel Bracha. An o(logn) expected rounds randomized byzantine generals protocol. In *Proceedings of the ACM Symposium on the Theory of Computation(STOC)*, 1985.

[9] Jason Cooper and Nathan Linial. Fast perfect-information leader-election protocol with linear immunity. *Combinatorica*, 15:319–332, 1995.

[10] D. Dolev, M. Fischer, R. Fowler, N. Lynch, and H. Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 1982.

[11] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. *SIAM Journal on Computing*, 17:975–988, 1988.

[12] Uriel Feige. Noncryptographic selection protocols. In *Proceedings of 40th IEEE Foundations of Computer Science(FOCS)*, 1999.

[13] J. Kahn, G. Kalai, and N. Linial. The influence of random variables on boolean functions. In *Proceedings of 29th IEEE Foundations of Computer Science(FOCS)*, 1988.

[14] N. Linial. Games computers play: Game-theoretic aspects of computing. Technical report, Hebrew University of Jerusalem, 1992.

[15] R. Ostrovsky, S. Rajagoplan, and U. Vazirani. Simple and efficient leader election in the full information model. In *Proceedings of the Twenty-Sixth Annual ACM Syposium on Theory of Computing*, 1994.

[16] A. Russell, M. Saks, and D. Zuckerman. Lower bounds for leader election and collective coin-flipping in the perfect information model. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, 1999.

[17] A. Russell and D. Zuckerman. Perfect information leader election in log*n + o(1) rounds. In *Proceedings of 39th Annual Symposium on Foundations of Computer Science(FOCS)*, 1998.

[18] Michael Saks. A robust noncryptographic protocol for collective coin flipping. *SIAM Journal of Discrete Mathematics*, pages 240–244, 1989.

[19] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11:345–367, 1997.