Towards Secure and Scalable Computation in Peer-to-Peer Networks

Valerie King * Jared Saia † Vishal Sanwalani ‡ Erik Vee §

Abstract

We consider the problems of Byzantine Agreement and Leader Election, where a constant fraction $b < 1/3$ of processors are controlled by a malicious adversary. The first problem requires that all uncorrupted processors come to an agreement on a bit initially held by one of the uncorrupted processors; the second requires that the uncorrupted processors choose a leader who is uncorrupted.

Motivated by the need for robust and scalable computation in peer-to-peer networks, we design the first scalable *protocols for these problems for a network whose degree is polylogarithmic in its size. By scalable, we mean that each uncorrupted processor sends and processes a number of bits that is only polylogarithmic in n. (We assume no limit on the number of messages sent by corrupted processors.) With high probability, our Byzantine Agreement protocol results in agreement among a $1 - O(1/\ln n)$ fraction of the uncorrupted processors. With constant probability, our Leader Election protocol elects an uncorrupted leader and ensures that a $1 - O(1/\ln n)$ fraction of the uncorrupt processors know this leader.*

We assume a full information model. Thus, the adversary is assumed to have unlimited computational power and has access to all communications, but does not have access to processors' private random bits.

1 Introduction

The potential of peer-to-peer(p2p) networks has been hobbled by our lack of understanding of how to design robust algorithms for large-scale networks. Networks that are truly p2p are currently very unsophisticated computationally: they are only able to provide the functionality of very basic data structures such as hash tables or skip lists. This limits their use to problems of storing, retrieving and disseminating data.

This lack of sophistication is in spite of strong motivation for building p2p networks that do non-trivial computation. For example, the company Cloudmark maintains a hybrid peer-to-peer network for detecting spam that is already being used by over one hundred million people [28]. Users in this network mark those emails they receive that they consider spam; fingerprints are then generated for emails that are marked as spam by many users and these fingerprints are distributed to all users in the network. The network is *hybrid* peer-to-peer in that it critically relies on a central authority that tallies votes and performs admission control by charging users an annual fee in order to participate in the network. It would benefit many people financially if there were a truly peer-to-peer network for spam prevention that did not charge a participation fee. However, designing such a network requires the development of fully distributed algorithms that can function robustly without admission control. These distributed algorithms would need to work correctly even in the case where an adversary controls a large number of the peers in the network. E.g., the algorithms would need to be robust to attacks by "zombienets" i.e. large numbers of computers that have been compromised and are now under the control of a single malicious user.

We note that peer-to-peer algorithms have been proposed not just for spam detection but also for a wide variety of other computationally challenging tasks including: worm detection and suppression [27, 23], collaborative filtering [9], reputation management [21, 1] and data mining [11]. Unfortunately, all algorithms proposed for these problems have no theoretical guarantees of being able to work in a network where an adversary controls a large number of the peers. This is a barrier to implementation and wide-spread use of these algorithms.

In this paper, we take a step towards increasing our understanding of how to design robust algorithms for large-scale peer-to-peer networks. In particular, we focus on designing robust and scalable algorithms for two fundamental problems: leader election and Byzantine agreement. Solutions to these two problems can be used as building blocks for robust and scalable solutions to other problems. For example, Byzantine agreement is a key component of al-

*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@cs.unm.edu. This research was partially supported by NSF grant CCR-0313160 and Sandia University Research Program grant No. 191445.

‡Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON; email: vishal@cs.unm.edu

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

gorithms for voting; database management [17, 18]; and secure multiparty computation [29, 19, 3, 4]. Our algorithms have the following important properties.

- *Robustness:* Our algorithms work on networks where strictly less than a $1/3$ fraction of the nodes are controlled by an adversary (i.e. suffer Byzantine faults)

- *Scalability:* The latency of our algorithms and the number of bits that each uncorrupted peer sends and processes are at most polylogarithmic in n, the number of peers in the network. Moreover our algorithms operate on networks where each peer has a number of neighbors that is at most polylogarithmic in n.

- *Almost-Everywhere Agreement:* It is easy to see that in a sparse network, an adversary can isolate some number of uncorrupted processors by taking over all their neighbors. Hence, our algorithms only guarantee that all but a $O(1/\ln n)$ fraction of the correct processors terminate with the correct output.

The problem of distributed computing with a malicious adversary in a sparse network was first considered in Dwork et. al. [14] in 1988 and improved in [7, 8, 26, 6]. All of these results require that each processor use a polynomial number of bits of communication. Our contribution is to bring the number of bits of communication down to polylogarithmic number of bits per processor; the computation time spent at each processor is polylogarithmic as well if we assume that each processor knows the topology of the network at the start. Our protocols are randomized; for Byzantine agreement the protocol succeeds with high probability. With leader election, success is achieved with a constant probability. Our communication model and our model of attack are the same as in the full information model described in the 2006 paper by Ben-Or, et. al. [5], except that uncorrupted processors can only communicate along the fixed edges of the network.

1.1 Our model We design a network with n processors where each node has degree that is polylogarithmic in n. We assume that communication among uncorrupted processors can only occur over edges in the network. Communication occurs in *rounds*. In each round, every processor may send out messages to its neighbors in the network. The corrupted processors are assumed to have received the messages of all the uncorrupted processors before they send out 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. The size of each message is polylogarithmic in n.

We assume the adversary gets to pick which processors will be corrupted before the algorithm begins and controls the actions of all corrupted processors so as to maximize the chance of causing the algorithm to fail. Also the adversary is computationally unbounded which disallows the use of cryptographic assumptions. We assume that each processor has access to private random bits which are not known to other processors or the adversary. All other information, e.g. the topology of the network, the algorithms run by the uncorrupted nodes, the initial inputs of the uncorrupted nodes etc., is assumed to be known by the adversary.

1.2 Our Results In this paper, we relax the requirement that *all* uncorrupted processors reach agreement as to the leader (or the value of bit in the case of Byzantine Agreement) at the end of the protocol, instead requiring that a $1 - o(1)$ fraction of uncorrupted processors reach agreement. This relaxation is sometimes referred to as almost-everywhere agreement. Our main results are stated in Theorems 1.1 and 1.2 below.

THEOREM 1.1. *Suppose there are n processors and a constant fraction, $b < 1/3$ of these processors are corrupted. There is a polylogarithmic (in n) bounded degree network and a protocol such that*

- *With constant probability greater than 0, a $1 - O(1/\ln n)$ fraction of the uncorrupted processors agree on a leader.*

- *Every uncorrupted processor sends and processes only a polylogarithmic (in n) number of bits.*

- *The number of rounds required is polylogarithmic in n.*

THEOREM 1.2. *Suppose there are n processors and a constant fraction $b < 1/3$ of these processors are corrupted. There is a polylogarithmic (in n) bounded degree network and a protocol such that:*

- *With high probability, a $1 - O(1/\ln n)$ fraction of the uncorrupted processors agree on the same value (for the bit).*

- *Every uncorrupted processor sends and processes only a polylogarithmic (in n) number of bits.*

- *The number of rounds required is polylogarithmic in n.*

The algorithms we use to prove Theorem 1.1 and 1.2 are adaptations of the Scalable Leader Election algorithm described in [22] for a fully connected network.

1.3 Related Work There has been significant work in designing peer-to-peer networks that are provably robust to a large number of Byzantine faults [16, 24, 20, 2, 25]. This work focuses only on robustly enabling storage and retrieval of data items. To the best of our knowledge, there is no previous work in the peer-to-peer literature that focuses on provably robust algorithms for more sophisticated computation.

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. [14] on fault tolerance in bounded degree networks. They exhibited a communication network of bounded degree connecting n processors that enables all but $O(t)$ correct processors to reach agreement in the presence of up to $t = O(n/\ln n)$ faulty processors. Berman and Garay improved on the efficiency of these protocols [7, 8]. Their main result is an algorithm that achieves consensus in the butterfly network using $O(t+\ln n \ln\ln n)$ one-bit parallel transmission steps, while tolerating $t = O(n/\ln n)$ corrupted processors and having $O(t \ln t)$ confused processors (i.e. uncorrupted processors that have decided on the incorrect bit). The number of rounds, number of corrupted processors that can be tolerated, and number of confused processors in this result are all asymptotically optimal for the butterfly network. Upfal proved that for any strong-enough expander, there is an almost everywhere agreement protocol that tolerates linearly many faulty processors [26]. The local computation required by each processor is exponential in his protocol even though the communication complexity is polynomial. Ben-Or and Ron designed a bounded degree network and an almost-everywhere agreement algorithm that is fully polynomial and tolerates a linear number of faults with high probability if the faulty processors are randomly located throughout the network [6].

Dolev and Reischuk show a lower bound on deterministic communication complexity of Byzantine agreement [13]. They show that each processor must send $\Omega(n)$ bits.

In [22], King et al. describe protocols for Leader Election and Byzantine Agreement that take polylogarithmic rounds and require each processor to send and process a polylogarithmic number of bits. These protocols only run on fully connected networks. Adapting these protocols to sparse networks is challenging because of problems with load-balancing. For example, the idea of simulating a scalable leader election protocols on a standard expander-based network topology such as those in [16, 24, 2, 26] fails to work. The problem is that the adversary can wait until near the end of the computation, identify those nodes that play a critical role in completing the computation and then send a large number of messages through the network to neighbors of those nodes. This type of denial of service attack allows the adversary to dynamically cut off important nodes from the rest of the network by overloading their neighbors with messages. This is a valid attack in our model since we assume the corrupt processors can send out an arbitrary number of messages.

To avoid this attack, we introduce a new way of dynamically updating permissible communication paths. Each processor dynamically updates, during the computation, those processors to which it will listen and those processors for which it will forward messages. This protects most processors from being overloaded by spurious messages.

2 Overview

Throughout the paper we use c to represent a (sometimes different) constant. Sometimes we specify the value of c; if its value is unspecified c should be read as representing a sufficiently large constant. We 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 $c > 3$. For readability, we treat $\ln n$ as an integer.

We now give a high-level sketch of how our protocol works. In its simplest form as in [22], 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 recursively repeat this step on the set of elected processors until there is a set of processors left which is polylogarithmic in size. At this point, the remaining processors in this set run either a known (randomized) Leader Election protocol or (deterministic) Byzantine Agreement protocol. Provided the fraction of corrupted processors is bounded away from $1/3$ (which we insure w.h.p.) then either with constant probability these processors (in the set) elect a uncorrupted leader or with probability 1, they achieve Byzantine Agreement. We then show how these processors can communicate the leader (or a bit value) to the rest of the processors.

In [22], the groups of processors are viewed as nodes in a layered network. There, since the processors are assumed to be fully connected, processors that are "elected" can directly communicate with each other. In this paper, we will refer to the network of [22] as an *election graph*, and we will show how it may be implemented robustly with a polylogarithmically bounded *static network* to achieve our results.

For the convenience of the reader, we describe the election graph in Section 3. We then describe our static network in Section 4. Section 6 contains the protocols which establish Theorems 1.1 and 1.2, we establish their correctness in Section 7.

Our protocol makes use of the Byzantine Agreement

protocol in [12], which takes $O(k)$ rounds and requires $O(k^3)$ bits of communication when there are k processors, provided the fraction of corrupted processors is bounded above by $1/3$. (We note that a straightforward modification of the protocol described in [22] gives a Byzantine Agreement protocol which w.h.p. (in k) works in $O(\ln k)$ rounds and uses $O(k^2)$ bits of communication, so long as the fraction of corrupted processors is strictly less than $1/3$. Although this does not qualitatively change our results, it results in a significantly faster protocol in practice.) We refer to this protocol as HEAVYWEIGHT-BYZANTINE-AGREEMENT. We implement this protocol on groups of k processors for $k = O(\ln^{10} n)$. Our protocols also make use of two protocols which we call HEAVYWEIGHT-LEADER-ELECTION and ELECT-SUBCOMMITTEE. These protocols, described in [22] allow small groups of processors to elect a leader or a subcommittee respectively. These protocols adapt protocols from [15] which are designed for the broadcast model, by simulating broadcast when necessary. In particular, each time it is necessary for processor p to broadcast in the original protocol, we simulate broadcast by having p send the message to all other processors followed by a call to HEAVYWEIGHT-BYZANTINE-AGREEMENT ensuring all the processors agree on the message. Although Byzantine Agreement protocols are designed for a single bit, it is easy to see that they can also be used to agree on a b-bit message for $b > 1$ as well; simply run the protocol in parallel b times, once for each bit.

The HEAVYWEIGHT-LEADER-ELECTION protocol is a straightforward adaptation of the protocol from [15], so we omit its description here. We give a high level sketch of the ELECT-SUBCOMMITTEE protocol below. A more detailed description will be given in Section 6. We assume the processors know each others' identities at the start.

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

1 For $i = 1$ to k,
2 Processor p_i randomly selects one of $k/(c \ln^3 n)$ "bins" and tells the other processors in its committee which bin it has selected.
3 The other processors in the committee run HEAVYWEIGHT-BYZANTINE-AGREEMENT to come to a consensus on which bin p_i has selected.
4 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.
5 Add the $|S_B| - c \ln^3 n$ lowest numbered processors in the input which are not in S_B to S_B to ensure $|S_B| = c \ln^3 n$.
6 Return S_B as the elected subcommittee.

LEMMA 2.1. *Let S be a committee of $\Omega(\ln^8 n)$ processors, where the fraction, f_S, of uncorrupted processors is greater than $2/3$. Then there exists some constant c, such that w.h.p., the subcommittee election protocol elects a subset Z of S such that $|Z| = c \ln^3 n$ and the fraction of uncorrupted processors in Z is greater than $(1 - 1/\ln n) f_S$. This protocol uses a polylogarithmic number of bits and polylogarithmic number of rounds in a fully connected network.*

The proof of Lemma 2.1 is a simple adaptation of a proof in [15]. The proof follows from a straightforward application of Chernoff bounds to show that with high probability any bin is selected by a number of uncorrupted processors that is not far below the expected number. A union bound shows that with high probability this holds for all bins.

3 The election graph

Our algorithms make use of an election graph to determine which processors will participate in which elections. This graph was described in [22] and is repeated here.

Before describing the election graph, we first present a result similar to that used in [10]. 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

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

In other words, $\mathcal{F}(X', \delta)$ is the set of all subsets of $\mathcal{F}$ whose overlap with X' is larger than the "expected" size by more than a δ fraction.

Let $\Gamma(r)$ denote the neighbors of node r in a graph.

LEMMA 3.1. *Let l, r, z, n be positive integers such that l and r are all no more than n and $r/l \geq ln^{1-z}n$. Then, there is a bipartite graph $G(L, R)$ such that $|L| = l$ and $|R| = r$ and*

1. *Each node in R has degree $\ln^z n$.*
2. *Each node in L has degree $O((r/l) \ln^z n)$.*
3. *Let $\mathcal{F}$ be the collection of sets $\Gamma(r)$ for each $r \in R$. Then for any subset L' of L, $|\mathcal{F}(L', 1/\ln n)| < \max(l, r)/\ln^{z-2} n$.*

The proof of Lemma 3.1 follows from a counting argument using the probabilistic method and is omitted.

The following corollaries follows immediately by repeated application of the above lemma.

COROLLARY 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 $G(L_i, R_i), i = 0, 1, \ldots, \ell^*$, and constants c_1 and c_2 such that $|L_i| = n/\ln^i n$, $|R_i| = n/\ln^{i+1} n$, and*

1. *Each node in R_i has degree $\ln^{c_1} n$.*
2. *Each node in L_i has degree $O(\ln^{c_2} n)$.*
3. *Let $\mathcal{F}$ be the collection of sets $\Gamma(r)$ for each $r \in R$. Then for any subset L'_i of L_i, $|\mathcal{F}(L'_i, 1/\ln n)| < |R_i|/\ln^6 n$.*
4. *Let $\mathcal{F}'$ be the collection of sets $\Gamma(l)$ for each $l \in L$. Then for any subset R'_i of R_i, $|\mathcal{F}'(R'_i, 1/\ln n)| < |L_i|/\ln^6 n$.*

COROLLARY 3.2. *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. *Let $\mathcal{F}$ be the collection of sets $\Gamma(r)$ for each $r \in R$. Then for any subset L'_i of L_i, $|\mathcal{F}(L'_i, 1/\ln n)| < |L_i|/\ln^3 n$.*

Lemma 3.1 and its corollaries show there exists a family of bipartite graphs with strong expansion properties which allow the formation of subsets of processors where all but a small fraction contain a majority that are uncorrupted.

We are now ready to describe the election graph. Throughout, we refer to nodes of the election graph as *e-nodes* to distinguish them from nodes of the static 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 e-node which is adjacent to every e-node in layer $\ell^* - 1$. For the remaining layers $\ell = 0, 1, ..., \ell^* - 1$, there are $n/\ln^{\ell+1} n$ e-nodes. There is an edge between the ith e-node, A, in layer ℓ and the jth e-node, B, 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 Corollary 3.2. In such a case, we say that B is the parent of A, and A is the child of B. Note that e-nodes have many parents.

Each e-node will contain a set of processors known as a *committee*. All e-nodes, except for the one on the top layer and those in layer 0, will contain $c\ln^3 n$ processors. Initially, we assign the n processors to e-nodes on layer 0 using the bipartite graph $G(L_0, R_0)$ described in Corollary 3.2. The i^{th} processor is a member of the committee contained in the j^{th} e-node of layer 0 iff there is an edge in G between the i^{th} node of L_0 and the j^{th} node of R_0. Note every e-node on layer 0 has $\ln^5 n$ processors in it.

The e-nodes on higher layers have committees assigned to them during the course of the protocol. Let A be an e-node on layer $\ell > 0$, let $B_1, \ldots, B_s$ be the children of A on layer $\ell - 1$, and suppose that we have already assigned committees to e-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^*, the processors in $A, B_1, \ldots, B_s$, run HEAVYWEIGHT-LEADER-ELECTION for Leader Election or HEAVYWEIGHT-BYZANTINE-AGREEMENT for Byzantine Agreement.

4 The polylogarithmic bounded degree static network.

We now describe the bounded degree static network and show how it can be used to hold elections specified by the election graph. For each e-node A, we form a collection of processors which we call the *s-node* $s(A)$. Intuitively, the s-node $s(A)$ serves as a central communication point for an election occurring at e-node A. Our goal is to bound the fraction of s-nodes controlled by the adversary by a decreasing function in n, namely $1/\ln^{10} n$, for each layer. As the number of s-nodes grows much smaller with each layer, we need to make each s-node more robust. To do this, the number of processors contained in the s-node increases with the layer. Specifically, the s-nodes for layer i are sets of $\ln^{i+12} n$ processors. We determine these s-nodes using the bipartite graph from Lemma 3.1, where L is a collection of n nodes, one for each processor, R is the set of s-nodes for layer i and the degree of each node in R is set to $\ln^{i+12} n$. The neighbors of each node in R constitute a set of processors in an s-node on layer i.

We use the term *link* to denote a direct connection in the static network. Since all the processors cannot communicate directly with each other, the communications for an election A will all be routed through $s(A)$: a message from a processor x to $s(A)$ on layer i will pass from the processor to a layer 0 s-node, whose processors will forward the message to a layer 1 s-node and so on, the goal being to reliably transmit the message via increasingly larger s-nodes up to $s(A)$. Similarly, communications to an individual processor x from $s(A)$ will be transmitted down to a layer 0 s-node whose processors will transmit the message to x. We describe the connections in the static network.

Connections in the static network

- Let A be an e-node on layer 0 in the election graph. Every processor in A has a link to every processor in $s(A)$.
- Let A and B be e-nodes in the election graph at levels i and $i - 1$ respectively such that A is a parent of B. Thus, $s(A)$ has $\ln^{i+12} n$ processors in it and $s(B)$ has $\ln^{i+11} n$ processors in it. Let G be a bipartite graph as

in Lemma 3.1 where L is the set of processors in $s(A)$, R is the set of processors in $s(B)$ and the degree of R is set to $\ln^{c_1} n$ and the degree of L is set to $O(\ln^{c_2} n)$. If there is an edge between two nodes in L and R respectively, then the corresponding processor in $s(A)$ has a link to the corresponding processor in $s(B)$. We will sometimes say that $s(A)$ is adjacent to $s(B)$ in the static network.

The following lemma follows easily from the application of Lemma 3.1 and its corollaries. Item (1) follows from Lemma 3.1; items (2) and (4) from Corollary 3.2; and item (3) from Corollary 3.1. Although item (2) only makes a guarantee about layer 0 e-nodes, we will see eventually that w.h.p., the fraction of bad e-nodes on every layer is small.

LEMMA 4.1. *W.h.p., the election graph and the static network have the following properties:*

1. *(Bad s-nodes) Any s-node whose fraction of corrupt processors exceeds $b+1/\ln n$ will be called bad. Else we will call the s-node good. No more than a $1/\ln^{10} n$ fraction of s-nodes on any given layer are bad.*
2. *(Bad e-nodes) Any e-node whose fraction of corrupt processors exceeds $b+1/\ln n$ will be called bad. Else we call the e-node good. No more than a $1/\ln^2 n$ fraction of e-nodes on layer 0 are bad.*
3. *(Bad s-node to s-node connection) For any pair of e-nodes A and B joined in the election graph, the processors in s-nodes $s(A)$ and $s(B)$ are linked such that the following holds. For any subset S of processors in $s(A)$, at most a $1/\ln^6 n$ fraction of processors in $s(B)$ have more than a $|S|/|s(A)| + 1/\ln n$ fraction of their links to $s(A)$ with processors in S.*
4. *(Bad e-node to e-node connection) Let $|I|$ represent the total number of e-nodes on layer i in the election graph. For any set S of e-nodes on any layer i, at most a $1/\ln^2 n$ fraction of e-nodes on layer $i+1$ have more than $|S|/|I|+1/\ln n$ fraction of their neighbors in S.*

OBSERVATION 4.1. *The degree of the static network is polylogarithmic.*

5 Communication Protocols

A *permissible path* is a path of the form $P = x, s(A_0), s(A_1), ..., .s(A_i)$ where x is a processor in A_0, i is the current layer of elections being held, each A_j is an e-node on layer j, and there is an edge in the election graph between A_j and A_{j+1} for $j = 0, ..., i$. Each processor y in an s-node $s(A)$ on each layer j keeps a *List* of permissible paths which determine which processors' messages it will forward. The List (for $y \in s(A)$) represents y's view of which processors are elected (to the subcommittee) at A that are still participating in elections on higher layers. If y's List indicates that x is such a processor, then the List will also have the entire path for x, which stretches from x to the elections on layer i in which x is currently participating in. We have the following definitions.

- We say a *s-node knows a message* [resp., *knows a permissible path*, or resp., *knows a* List *of permissible paths*] if $1 - b - 2/\ln n$ processors in the s-node are uncorrupted and receive the same message [resp., have the same path on their Lists, or resp., all have the same List.]
- A permissible path P is good if every s-node on the path knows P. Else the path is bad. We will show our construction of the static network ensures at most a $1/\ln n$ fraction of the permissible paths are bad.

We now describe three primitive communication subroutines: SENDHOP, SEND, and MESSAGEPASS. The subroutine SENDHOP describes how s-nodes (with direct links) communicate with each other, SEND describes how a processor communicates with an s-node, and MESSAGEPASS describes how two processors communicate with each other.

SENDHOP(s, r, m, P): A message m can be passed from s (the sender) to r (the receiver) from a level i to a level $i-1$ or from a level i to a level $i+1$, where s and r are s-nodes on these layers or one of s, r is a 0-layer s-node and the other is a processor. If a processor x sends a message to a layer 0 s-node $s(A)$ it sends the message to every processor in $s(A)$ (note by construction it will have a direct link with every processor in $s(A)$). Similarly if a message is sent from a layer 0 s-node $s(A)$ to a processor x, every processor in $s(A)$ sends the message to x.

When an s-node $s(A)$ sends a message to s-node $s(B)$, every processor in $s(A)$ sends the message to those processors of $s(B)$ to which it has a direct link. When each processor in $s(B)$ receives such a set of messages, it takes the majority to determine the message. If there is no majority value, the processor ignores the message. Along with sending the message the processors also send information which specifies along which path P the message is being sent. Each time a message is received by a processor of an s-node $s(B)$ on layer $j \leq i$, it checks that

1. The message came from the s-node previous to it in the path P; if not the message is dropped.
2. The path P (or its reverse) is on its List of permissible paths. If not, the message is dropped.

3. Only messages that conform to the protocol in size and number are forwarded up and down the permissible paths. If more or longer messages are received from a processor, messages from that processor are dropped.

SEND(s, r, m, P): {Of the first two parameters, one must be a processor ("x") and one must be an s-node ("$s(A)$"). The path P contains the first parameter s as its start and the second parameter r as its endpoint.} SEND(s, r, m, P) sends the message m from s to r along the path P via repeated application of SENDHOP.

MESSAGEPASS$(x \in A, y \in B, m, P_x, P_y)$: { Both A and B are adjacent e-nodes. Hence, $s(A)$ and $s(B)$ are adjacent in the static network. } A message from processor x in e-node A sends message m to processor y in e-node B by first calling SEND$(x, s(A), m, P_x)$. Then $s(A)$ sends m to $s(B)$ by calling SENDHOP$(s(A), s(B), m, P)$, where P is the path consisting of two s-nodes $s(A), s(B)$. Finally, the message is transmitted from $s(B)$ to y by calling SEND$(s(B), y, m, P_y^r)$, where P_y^r is the reversal of path P_y.

6 The protocols for LEADER-ELECTION and BYZANTINE-AGREEMENT

Before we describe the LEADER-ELECTION and BYZANTINE-AGREEMENT protocols, we first need to adapt the ELECT-SUBCOMMITTEE protocol for the static network. Let A be an e-node with children $B_1, \ldots, B_s$, and let X be the set of all processors from $B_1, \ldots, B_s$. For each $i \in [s]$ and $x \in B_i$, let P_x denote a good path of s-nodes from x to $s(B_i)$, concatenated with $s(A)$. At the start of the election for A, we assume that each node in P_x knows P_x and $s(A)$ knows $\{P_x \mid x \in X\}$. We now describe the implementation of the subcommittee election. Every processor $x \in X$ sends a random bin number m_x to every other processor $y \in X$. The processors use the HEAVYWEIGHT-BYZANTINE-AGREEMENT protocol to determine a consensus on the bin numbers sent by each processor. (Recall that the number of processors in e-nodes is always polylogarithmic, so this can be done sending only polylogarithmic messages.) The bin numbers are sent up to $s(A)$, where each processor in $s(A)$ takes a majority to determine the bin numbers, from which it computes the lightest bin and the winners. Then $s(A)$ sends down the list of winners along all the permissible paths to each processor $x \in X$. Processors on the path (i.e. in s-nodes along the path) update their Lists of permissible paths to remove those processor-paths who lost as well as those processor-pairs who won too many elections (we will quantify this shortly), and make $\ln^4 n$ copies of each of the winners' paths and concatenate a different layer $i+1$ s-node parent onto each one. We present a detailed description of ELECT-SUBCOMMITTEE below.

ELECT-SUBCOMMITTEE:

1 For each $x \in X$: *// This phase done in parallel*
  Processor x randomly selects one of $k/(c_1 \ln^3 n)$ "bins". Let m_x be the bin selected. Processor x tells every other processor $y \in X$ its bin selection m_x using MESSAGEPASS$(x, y, m, P_x, P_y,)$.
2 The processors of X run HEAVYWEIGHT-BYZANTINE-AGREEMENT to come to a consensus on m_x.
3 Each $y \in X$ sends m_x to $s(A)$ by calling SEND$(y, s(A), m_x, P_y)$.
4 The processors in $s(A)$ determine each m_x by taking the majority of messages they receive. Each determines the processors in the lightest bin, adding the lowest numbered processors if needed to bring the number to $c \ln^3 n$. These become the elected processors.
5 For each processor $x \in X$ that is elected, the processors in $s(A)$ use SEND$(s(A), x, m, P_x^r)$ to tell x, along with each s-node in P_x, that x was elected.
6 Each processor in each s-node revises its List of permissible paths to:
  Retain only the winners.
  Eliminate processors who have won more than 8 elections.
  Make $\ln^4 n$ copies of remaining permissive paths, concatenating each with a different s-node neighbor on layer $i+1$.
7 $s(A)$ sends its List to every adjacent s-node $s(B)$ on layer $i+1$ using SENDHOP$(s(A), s(B), m, P)$, where P is the path consisting only of $s(A), s(B)$.

The condition in Step 5 that requires processors who have won more than 8 elections to be eliminated is a technical condition that insures the protocol is load-balanced and processors in an s-node do not communicate more than a polylogarithmic number of bits. We now describe the Leader Election (Byzantine Agreement) protocol.

LEADER-ELECTION [resp., BYZANTINE-AGREEMENT]:

1 For $\ell = 1$ to ℓ^*:
2 For each e-node A in layer ℓ, (Let $B_1, \ldots, B_s$ be the children of A in layer $\ell - 1$ of the election graph.)
3 If $\ell < \ell^*$, run ELECT-SUBCOMMITTEE on the processors in nodes $B_1, \ldots, B_s$. Assign winning processors to node A.
4 Else, run HEAVYWEIGHT-LEADER-ELECTION [resp., HEAVYWEIGHT-BYZANTINE-AGREEMENT] on the processors in nodes $B_1, \ldots, B_s$.
5 Let A^* be the e-node on layer ℓ^*, every processor x assigned to A^* communicates the result of

HEAVYWEIGHT-LEADER-ELECTION
[resp., HEAVYWEIGHT-BYZANTINE-AGREEMENT]
to $s(A^*)$ using SEND$(x, s(A^*), m, P_x)$.

6 Every processor in $s(A^*)$ takes the majority to determine the result of HEAVYWEIGHT-LEADER-ELECTION [resp., HEAVYWEIGHT-BYZANTINE-AGREEMENT].

Note every processor is a member of $s(A^*)$, thus Steps 5 and 6 will insure the final result of the protocol is communicated to every processor.

7 Proofs

To establish the correctness of the protocol presented in Section 6, we first state some claims regarding the primitive communication protocols. Their proofs follow by straightforward probabilistic arguments and are omitted in the interest of space. Recall the fraction of corrupted processors is b, where $b < 1/3$.

CLAIM 7.1. *Let $s(A)$ and $s(B)$ be s-nodes on consecutive layers. Assume the following three conditions hold.*

- *Both $s(A)$ and $s(B)$ are good.*
- *$s(A)$ is on a permissible path known by $s(B)$.*
- *There exists a set S of processors from $s(A)$ such that for every message m, all processors in S are uncorrupted and agree on a message m. Further S consists of at least a $1 - b - 2/\ln n$ fraction of the processors in $s(A)$*

Then there is a set S' of processors from $s(B)$ such that for every message m, every processor in S' is uncorrupted and agrees on the message m after SENDHOP$(s(A), s(B), m, P)$ *is called. (Here, P is the path $s(A), s(B)$.) Further, S' consists of all but a $1/\ln^6 n$ fraction of the uncorrupted processors in $s(B)$.*

CLAIM 7.2. *Let x be an uncorrupted processor. Assume P_x is a good path. Then after* SEND$(x, s(A), m, P_x)$ *is executed, there is a fixed set S of uncorrupted processors which contains all but a $1/\ln^6 n$ fraction of the uncorrupted processors in $s(A)$ and every processor $z \in S$ agrees on m.*

An election at e-node A is *legitimate* if the following two conditions hold simultaneously for more than a $2/3$ fraction of processors x participating in the election at A: (1) Processor x is uncorrupted; (2) The path P_x is good.

The following lemma shows when the election at e-node A is *legitimate*, all uncorrupted processors that have good paths to $s(A)$ come to agreement about the List of winners, and this List is known by all s-nodes on their paths.

LEMMA 7.1. *For a legitimate election at node A, let X be a set of uncorrupted processors with good permissible paths. (Note $|X| > 2\ln^8 n/3$.) Let S be the set of uncorrupted processors in $s(A)$ that know X. Then after the execution of* ELECT-SUBCOMMITTEE*, the processors in S know the winners of the election in A, as do the s-nodes that belong to good paths P_x.*

Proof. From Claim 7.2, we have that every message m sent by MESSAGEPASS$(y \in B_i, z \in B_j, m, P_y, P_z)$ from $y \in X$ to $z \in X$ is received by some fixed set S of uncorrupted processors in $s(B_i)$, such that S contains at a least $1 - b - 2/\ln n$ fraction of the processors in $s(B_i)$. By Claim 7.1, every message sent by y is received by z. Since X contains more than $2/3$ of the total processors participating in the election, (after running HEAVYWEIGHT-BYZANTINE-AGREEMENT) all the processors in X will all agree on the same set of bin values. Thus after the processors in X send these values to $s(A)$, $s(A)$ will know the winners. When $s(A)$ sends these winners to X, by repeated application of Claim 7.1, we have every $x \in X$ and every s-node in P_x will know these winners.

We have shown that in a legitimate election at node A, $s(A)$ knows the list of winners. We next consider when paths are dropped from the permissible path Lists.

7.1 The removal of permissible paths from Lists

Let y be a processor in some s-node on layer i. A permissible path P_x is removed from a processor y's List on layer i if y receives a message from an s-node above it in P_x, indicating either x has won more than 8 elections or x lost in the election held at the last node of P_x. Here, we consider when P_x is removed for the former reason. I.e., we give an upper bound on the fraction of processors that are reported to have won too many elections on layer i.

First we consider the effect of legitimate elections. The following lemma, a version of which appears in [22], shows that on a given layer a very small fraction of uncorrupted processors win more than 8 times in legitimate elections.

LEMMA 7.2. *W.h.p., the processors that win more than 8 elections, counting multiplicities, account for no more than a $16/\ln^3 n$ fraction of the uncorrupted processors that are winners of legitimate elections.*

Next we bound the effect of elections that are not legitimate. We first consider the case where $s(A)$ is good, yet the fraction of uncorrupted processors participating in A with good paths is less than 2/3. For the remainder of the proof we shall treat such an e-node A as a bad e-node.

CLAIM 7.3. *Suppose less than a $1/7$ fraction of the uncorrupted processors of a good $s(A)$ agree on a message*

m. *Then after* SENDHOP$(p(A), p(B), m, P)$ *is executed, all but a* $1/\ln^6 n$ *fraction of the good processors in* $s(B)$ *will ignore* m.

Proof. Even if the corrupted processors agree on m, since $b < 1/3$, the total fraction of processors in $s(A)$ sending the message m is less than 10/21. Thus at most a $1/\ln^6 n$ fraction of the processors in $s(B)$ will receive m from a majority of processors in $s(A)$.

Hence a good $s(A)$ can only communicate 7 different sets of winners to the s-nodes below it. Since each uncorrupted processor will send $\ln^3 n$ winners, the total number of winners sent is at most $7\ln^3 n$. Hence a bad e-node can cause at most $7\ln^3 n$ processors to have their permissible paths removed.

Next we consider the effect of a bad s-node. We will assume one bad s-node $s(A)$ on layer i can cause the removal of all the permissible paths for every processor participating in the election at A. Since $\ln^8 n$ processors participate in an election, and fewer than a $1/\ln^{10} n$ fraction of the s-nodes are bad on a layer, the fraction of uncorrupted winners affected is less than $1/\ln^2 n$. Thus we can bound the fraction of the uncorrupted winners on any layer i that have their permissible paths removed by $1/\ln^2 n + 1/\ln^3 n + 7\beta_i$; where β_i represents the fraction of bad e-nodes on layer i. Thus we have the following lemma.

LEMMA 7.3. *Assume the fraction of bad e-nodes on layer* i *is bounded by* $c/\ln^2 n$*, for some constant* c*. Then the fraction of uncorrupted winners that have their permissible paths removed on layer* i *is bounded by* $8c/\ln^2 n$.

7.2 Proof of Theorems 1.1 and 1.2 We now complete the proof of Theorems 1.1 and 1.2. They will follow from the lemma below.

LEMMA 7.4. *On layer* i*, w.h.p., at least a* $1 - 4/\ln^2 n$ *fraction of s-nodes* $s(A_j)$ *have the following properties:*

- $s(A_j)$ *is good.*
- *At least a* $1 - b - 4i/\ln n$ *fraction of the processors in node* A_j *are uncorrupted and have good paths to* $s(A_j)$ *(note this implies* $s(A_j)$ *knows this path). That is,* A_j *is a good e-node.*

Proof. We prove the lemma by induction. On all layers and particularly layer 0, only a $1/\ln^{10} n$ fraction of the s-nodes are bad. If $s(A)$ is good, then every processor in A has a good path to $s(A)$. Further by construction all but a $1/\ln^2 n$ fraction of the e-nodes on layer 0 consist of at least a $1 - b - 1/\ln n$ fraction of uncorrupted processors. So the lemma is true on layer 0.

Assume the lemma is true for layer i. Then a $1 - 4/\ln^2 n$ fraction of e-nodes are good, more specifically these e-nodes have at least a $1 - b - 4i/\ln n$ fraction of uncorrupted processors that have a good path to their corresponding s-node. Since the election is legitimate by Lemmas 2.1 and 7.1, w.h.p., after ELECT-SUBCOMMITTEE at least a $1-b-4i/\ln n-1/\ln n$ fraction of the processors elected are uncorrupted and have a good path to any good parent of their s-node. Thus at least a $1-b-(4i+1)/\ln n$ fraction of the processors elected at layer i are uncorrupted and have good paths to good parent s-nodes on layer $i+1$. By Lemma 7.3 this fraction is reduced by at most $32/\ln^2 n$. Thus at least a $1 - b - (4i + 2)/\ln n$ fraction of the processors elected at layer i are uncorrupted and have good paths to good parent s-nodes on layer $i + 1$. Since the fraction of bad s-nodes on layer $i+1$ is at most $1/\ln^{10} n$, by Corollary 3.2 at least a $1 - 1/\ln^2 n - 1/\ln^{10} n$ fraction of the e-nodes (and their corresponding s-nodes) are good on layer $i+1$, and have at least a $1-b-(4i+2)/\ln n-1/\ln n$ fraction of uncorrupted processors that have good paths to their corresponding s-nodes.

By Lemma 7.4, w.h.p. the layer ℓ^* e-node is good. Thus the processors in this e-node either reach Byzantine Agreement by running HEAVYWEIGHT-BYZANTINE-AGREEMENT or elect an uncorrupted leader with constant probability by running HEAVYWEIGHT-LEADER-ELECTION . Since all the processors are in the s-node (though they may appear multiple times) corresponding to A on ℓ^*, by Claim 7.2 all but a $O(1/\ln n)$ fraction of the good processors learn the final result. To prove the number of bits sent by each processor is polylogarithmic we note each processor is in a polylogarithmic number of e-nodes and s-nodes on each layer i, and participates in at most a polylogarithmic number of election on layer i. Since the number of layers is $O(\ln n)$ Theorems 1.1 and 1.2 follow.

8 Conclusion

We have presented scalable and robust protocols for solving Byzantine agreement and leader election. Our protocols are scalable in that they operate in a network of n nodes where each node in the network has a number of neighbors that is polylogarithmic in n. Further, the latency of the protocols are polylogarithmic in n, and each processor sends and processes a number of bits that is polylogarithmic in n. Our algorithms are robust in the sense that they work correctly even if an omniscient and computationally unbounded adversary controls a constant fraction of the nodes in the network. Many open problems remain including the following. First, the protocols described in this paper work on static networks. Can we adapt them to work on dynamic networks, which can grow, shrink, or have significant node turnover during the course of the

computation? Second, what types of computational problems can and can not be solved in our model of scalable and robust computation? Third, can we design scalable and robust protocol that work correctly on any sparse network with sufficiently good expansion? Alternatively, can we prove that such a result is not possible? Finally, can we design simplifications of our protocols or heuristics based on our protocols that can be deployed in an actual peer-to-peer network?

References

[1] Z. Abrams, R. McGrew, and S. Plotkin. Keeping peers honest in eigentrust. In *2nd Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[2] B. Awerbuch and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. In *Thirty-First Int. Colloquium on Automata, Languages, and Programming (ICALP)*, 2004.

[3] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth ACM Symposium on the Theory of Computing (STOC)*, 1993.

[4] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, 1994.

[5] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine agreement in the full-information model in O(log n) rounds. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, 2006.

[6] M. Ben-Or and D. Ron. Agreement in the presence of faults, on networks of bounded degree. *Information Processing Letters*, 1996.

[7] P. Berman and J. Garay. Asymptotically optimal distributed consensus. In *Proceedings ICALP 89 (16th International Colloquium on Automata, Languages and Programming)*, 1989.

[8] P. Berman and J. Garay. Fast consensus in networks of bounded degree. In *Fourth International Workshop on Distributed Algorithms*, 1990.

[9] J. Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy*, 2002.

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

[11] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. In *IEEE Internet Computing special issue on Distributed Data Mining*, 2005.

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

[13] D. Dolev and R. Reischuk. Bounds on information exchange for byzantine agreement. In *Proceedings of the first annual ACM symposium on Principles of distributed computing(PODC)*, 1982.

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

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

[16] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the Thirteenth ACM Symposium on Discrete Algorithms (SODA)*, 2002.

[17] H. Garcia-Molina, F. Pittelli, and S. Davidson. Is byzantine agreement useful in a distributed database? In *the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems*, 1984.

[18] H. Garcia-Molina, F. Pittelli, and S. Davidson. Applications of byzantine agreement in database systems. *ACM Transactions on Database Systems (TODS)*, 1986.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[20] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proceedings of the 17th International Symposium on Distributed Computing*, 2004.

[21] S. Kamvar, M. Sclosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International World Wide Web Conference (WWW)*, 2003.

[22] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *Proceedings of 27th ACM-SIAM Symposium on Discrete Algorithms(SODA)*, 2006.

[23] D. Malan and M. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Third Workshop on Rapid Malcode (WORM)*, 2005.

[24] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[25] C. Scheideler. How to spread adversarial nodes? rotate! In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, 2005.

[26] E. Upfal. Tolerating linear number of faults in networks of bounded degree. In *10th Annual Symposium on Principles of Distributed Computing(PODC)*, 1992.

[27] V. Vlachos, S. Androutsellis-Theotokis, and D. Spinellis. Security applications of peer-to-peer networks. *Computer Networks*, 45:195–205, 2004.

[28] C. Website. http://cloudmark.com/.

[29] A. Yao. Protocols for Secure Computations. In *Proceedings of the Twenty-Third IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164, 1982.