Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary

Valerie King
Department of Computer Science, University of Victoria
and
Jared Saia
Department of Computer Science, University of New Mexico

We describe an algorithm for Byzantine agreement that is scalable in the sense that each processor sends only $\tilde{O}(\sqrt{n})$ bits, where n is the total number of processors. Our algorithm succeeds with high probability against an *adaptive adversary*, which can take over processors at any time during the protocol, up to the point of taking over arbitrarily close to a 1/3 fraction. We assume synchronous communication but a *rushing* adversary. Moreover, our algorithm works in the presence of flooding: processors controlled by the adversary can send out any number of messages. We assume the existence of private channels between all pairs of processors but make no other cryptographic assumptions. Finally, our algorithm has latency that is polylogarithmic in n. To the best of our knowledge, ours is the first algorithm to solve Byzantine agreement against an adaptive adversary, while requiring $o(n^2)$ total bits of communication.

Categories and Subject Descriptors: F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

General Terms: Theory

Additional Key Words and Phrases: Byzantine agreement, consensus, Samplers, Peer-to-peer, secret-sharing, Monte Carlo Algorithms, distributed computing

1. INTRODUCTION

Recent years have seen rapid growth in networks that are characterized by large sizes and little admission control. Such networks are open to attacks by malicious users, who may subvert the network for their own gain. To address this problem, the research community has been recently revisiting techniques for dealing with nodes under the control of a malicious adversary [Kotla et al. 2007; Clement et al. 2009; Clement et al. 2008; Anderson and Kubiatowicz 2002].

The Byzantine agreement problem, defined in 1982, is the *sine qua non* of handling malicious nodes. With a solution to Byzantine agreement, it is possible to create a network that is reliable, even when its components are not. Without a so-

Author's address: Valerie King, Department of Computer Science, P.O. Box 3055; Victoria, BC, Canada V8W 3P6. email: val@cs.uvic.ca. This research was partially supported by an NSERC grant.
Author's address: Jared Saia, Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386. email saia@cs.unm.edu. This research was partially supported by NSF CAREER Award 0644058, NSF CCR-0313160, and AFOSR MURI grant FA9550-07-1-0532.

lution, a compromised network cannot perform even the most basic computations reliably. A testament to the continued importance of the problem is its appearance in modern domains such as sensor networks [Shi et al. 2004]; mediation in game theory [Abraham et al. 2006; Abraham et al. 2008]; grid computing [Anderson and Kubiatowicz 2002]; peer-to-peer networks [Rhea et al. 2003]; and cloud computing [Wright 2009]. However, despite decades of work and thousands of papers, we still have no practical solution to Byzantine agreement for large networks. One impediment to practicality is suggested by the following quotes from recent systems papers (see also [Castro and Liskov 2002; Malkhi and Reiter 1997; Amir et al. 2006; Agbaria and Friedman 2003; Yoshino et al. 2005]):

— *"Unfortunately, Byzantine agreement requires a number of messages quadratic in the number of participants, so it is infeasible for use in synchronizing a large number of replicas"* [Rhea et al. 2003]

— *"Eventually batching cannot compensate for the quadratic number of messages [of Practical Byzantine Fault Tolerance (PBFT)]"* [Cowling et al. 2005]

— *"The communication overhead of Byzantine Agreement is inherently large"* [Cheng et al. 2009]

In this paper, we describe an algorithm for Byzantine agreement with only $\tilde{O}(n^{1/2})$ bit communication per processor overhead. Our techniques also lead to an algorithm with $\tilde{O}(n^{1/2})$ bit complexity which is a distributed version of a bit-fixing random source. That is, the algorithm generates a polylogarithmic length sequence consisting of mostly global coinflips generated uniformly and independently at random and agreed upon by all the good processors. Our algorithms are polylogarithmic in time and succeed with high probability.

Our algorithm has a small probability of error which is necessary to overcome the lower bound in [Dolev and Reischuk 1985]. That work shows that any deterministic algorithm using $o(n^2)$ messages in the synchronous model must err. Thus, any randomized algorithm which always uses $o(n^2)$ messages must necessarily err with positive probability, since the adversary can guess the random coinflips and cause the algorithm to err if the guess is correct. By way of comparison, we note that any algorithm that uses cryptography will also err with a certain probability, since the adversary may get lucky and break the cryptography.

1.1 Model and Problem Definition

We assume a fully connected network of n processors $p_1, p_2, ..., p_n$ whose ID's are common knowledge. Each processor has a private random coin. We assume that all communication channels are *private* and that whenever a processor sends a message directly to another, the identity of the sender is known to the recipient, but we otherwise make no cryptographic assumptions.

We assume an *adaptive* adversary. That is, the adversary can take over processors at any point during the protocol, up to a total of $1/3 - \epsilon$ fraction of the processors for any positive constant ϵ. When an adversary takes over a processor, it learns the processor's state. The adversary chooses the input bits of every processor, bad processors can engage in any kind of deviations from the protocol, including false messages and collusion, or crash failures, while the remaining processors are good and follow the protocol. Bad processors can send *any* number of messages. Finally, we assume the adversary is malicious and is actively trying to thwart the protocol run by the good processors.

We assume a synchronous model of communication. In particular, we assume there is a known upper bound on the transit time of any message and communication proceeds in rounds determined by this transit time. The time complexity of our protocols are given in the number of rounds. However, we assume a *rushing* adversary. That is we assume that the bad processors may wait to receive all messages sent by the good processors before they need to send out their own messages.

In the *Byzantine agreement* problem, each processor begins with either a 0 or 1. An execution of a protocol is *successful* if all processors terminate and, upon termination, agree on a bit held by at least one good processor at the start.

We define a variant of a bit-fixing (random) source, see [Chor et al. 1985], which we call a *distributed* (s, t) *random source* or (s, t) *random source*, for short. This produces a (s, t) *random sequence* of *words* $w_1, ..., w_s$ such that t are generated uniformly and independently at random, and are agreed upon by all good processors. An adversary chooses the indices of these t words and the values received by the processors of each remaining word (these values may not be agreed upon). In fixing the value(s) of a word at index j the adversary knows the values only of the words which appear in positions $i < j$. A simple algorithm to generate a $(n, 2n/3)$ random sequence which takes $O(n)$ time and $O(wn)$ bit complexity per processor for words of length w is as follows: For $i = 1, .., n$, each p_i randomly picks a word and sends it to all other processors. The challenge here is to reduce the length of the sequence and the communication cost per processor.

1.2 Results

We use the phrase *with high probability* (*w.h.p.*) to mean that an event happens with probability at least $1 - 1/n^c$ for every constant c and sufficiently large n. For readability, we treat $\log n$ as an integer throughout. In particular, we omit the floor or ceiling operation for $\log n$ when the necessary operation is clear from context.

In all of our results, n is the number of processors in a synchronous message passing model with an adaptive, rushing adversary that controls a less than $1/3 - \epsilon$ fraction of processors, for any positive constant ϵ. We have three main results. The first result makes use of the second and third ones, but these latter two results may be of independent interest. First, we show:

THEOREM 1. [*BA*] *There exists a protocol which w.h.p. computes Byzantine agreement, runs in polylogarithmic time, and uses $\tilde{O}(n^{1/2})$ bits of communication per processor.*

Our second result concerns *almost-everywhere* ("a.e.") Byzantine agreement, where a $1 - 1/\log n$ fraction of the good processors come to agreement on a good processor's input bit; and *almost-everywhere* ("a.e.") (s, t) random source, where a $1 - 1/\log n$ fraction of the good processors come to agreement on the t random words in the sequence of s words generated by the random source. For the definition of a.e. (s, t) random source, we allow the possibility that a different $1 - 1/\log n$ fraction of good processors agrees on each of the t random words in the sequence of s words generated.

THEOREM 2. [ALMOST EVERYWHERE BYZANTINE AGREEMENT] *For any $\delta > 4$, there exists a protocol that w.h.p. computes a.e. Byzantine agreement; uses $\tilde{O}(n^{4/\delta})$ bits of communication per processor; and runs in time $O((\log^{6+\delta} n/(\log\log n)^2)$. In addition, this protocol can be used as an a.e. $(s, 2s/3)$ random source for an additional cost of $O(\log n/\log\log n)$ time and $\tilde{O}(n^{4/\delta})$ bits of communication per bit of the sequence generated, for $s = \Omega(\log^{4+\delta} n)$.*

Our third result concerns going from a.e. Byzantine agreement to Byzantine agreement. It makes use of the a.e. (s, t) random source in Theorem 2. We actually prove a result below that is stronger than what is necessary to establish Theorem 1.

THEOREM 3. *Let c and ϵ be any positive constants. Assume $n/2 + \epsilon n$ good processors agree on a message m in a synchronous model with private channels and that we have an a.e. (s, t) random source that generates numbers between 1 and $\sqrt{n}$, with a latency of $f(s)$ per number and a cost of $g(s)$ bits per processor per number. Then, there is an algorithm that ensures with probability at least $1 - 1/n^c$ that all good processors output m. Moreover this algorithm runs in $O(sf(s))$ time and uses $\tilde{O}(s(n^{1/2} + g(s)))$ bits of communication per processor.*

1.3 Techniques

Our algorithm uses a sparse network construction and tournament tree similar to the network and tournament network in [King et al. 2006a]. This past result gives a bandwidth efficient a.e. Byzantine agreement algorithm for a *non-adaptive adversary*, which must take over all its processors at the start of the algorithm. The basic idea of the algorithm from [King et al. 2006a] is that processors compete in local elections in a tournament network, where the winners advance to the next highest level, until finally a small set is elected that is representative in the sense that the fraction of bad processors in this set is not much more than the fraction of bad processors in the general population. The local elections are held based on a variant of Feige's bin selection protocol [Feige 1999].

This approach is *prima facie* impossible with an adaptive adversary, which can simply wait until a small set is elected and then can take over all processors in that set. To avoid this problem, we make use of two novel techniques. First, instead of electing processors, we elect *arrays* of random numbers, each generated initially by a processor. Second, we use secret sharing on the contents of the arrays to make sure that: 1) the arrays are split among an increasingly larger numbers of processors as the array is elected higher up in the tournament; as the nodes higher in the tree have more processors (and the memories of previously held shares are erased), the adversary must corrupt an increasingly larger number of processors to learn the secret; 2) the secrets in the arrays cannot be reconstructed except at the appropriate time in the protocol. Critical to our approach is the need to iteratively reapply secret sharing on shares of secrets that were computed previously, in order to increase the number of processors the adversary must corrupt as elections occur higher up in the tournament.

Another contribution of this paper is a method *a.e.BA-with-random-source* for computing a.e. Byzantine Agreement given an a.e. random source. In [King et al. 2006a], each election was run by a small group of participants using Feige's bin selection protocol [Feige 1999]. Adapting Feige's bin selection protocol from an atomic broadcast model to a message-passing model requires a Byzantine agreement protocol. In [King et al. 2006a], this was run by the small group of election participants. Because we are now faced with an adaptive adversary which can adaptively corrupt small groups, we instead need to implement BA on a much larger sets of processors. To achieve this, we use our algorithm, *a.e.BA-with-random-source*, which is an a.e. version of Rabin's algorithm [Rabin 1983] (which requires a global coin), run on a sparse network. To run an a.e. version of Rabin's algorithm in a node on a particular level, we supply coinflips using an *a.e.random-source* generated from the arrays which won on the previous tournament level.

Our final new technique is a simple but non-obvious protocol for going from a.e. Byzantine agreement and an a.e. random source to Byzantine agreement, with an adaptive adversary (Section 4). A past result [King and Saia 2009] shows that it is possible to do this with a non-adaptive adversary, even without private channels. However, the technique presented in this paper for solving the problem with an adaptive adversary is significantly different from the approach from [King and Saia 2009].

1.4 Map of the algorithms

Initially, processor p_i provides an array of random bits to leaf i of the tournament tree. There are five algorithms: *BA*, *a.e.BA*, *a.e.BA-with-random-source*, *a.e.random-source*, and *a.e.BA-to-BA*.

—*BA* (Section 5) solves Byzantine agreement and yields Theorem 1.

—*BA* calls *a.e.BA* (Section 3) which solves a.e. Byzantine agreement, yielding Theorem 2, and then calls *a.e.BA-to-BA* (Section 4) which yields Theorem 3.

—*a.e.BA-to-BA* uses *a.e.random-source* to go from a.e. Byzantine agreement to Byzantine agreement.

—*a.e.random-source* (Section 3.7) is a simple modification of *a.e.BA* to provide an a.e. (s, t) random source.

—*a.e.BA* calls *a.e.BA-with-random-source* (Section 3.3) for each set of processors in each node of the tournament tree. This protocol produces a.e. Byzantine agreement for each set, given an a.e. random sequence for that set.

—Each call of *a.e.BA-with-random-source* uses an *a.e.random-source* created by the subtree of the node in question. This implementation of *a.e.random-source* is implicit in the description of *a.e.BA* in Section 3.

2. RELATED WORK

As mentioned previously, this paper builds on a main idea from [King et al. 2006a] which gives a polylogarithmic time protocol with polylogarithmic bits of communication per processor for a. e. Byzantine agreement, leader election, and universe reduction in the synchronous full information message passing model with a *non-adaptive* rushing adversary. The result from [King and Saia 2009] shows how the a.e. universe reduction problem (i.e., to select a small representative subset of processors known to almost all the processors) can be used to go from a.e. Byzantine agreement to Byzantine agreement with a *non-adaptive* adversary. Empirical results in [King et al. 2010] suggest that the algorithms from [King et al. 2006a;

King and Saia 2009] use less bandwidth in practice than other popular Byzantine agreement algorithms once the network size reaches about 1,000 nodes.

A.e. agreement in sparse networks has been studied since 1986. See [King et al. 2006a; 2006b] for references. The problem of almost everywhere agreement for secure multiparty computation on a partially connected network was defined and solved in 2008 in [Garay and Ostrovsky 2008], albeit with $\Omega(n^2)$ message cost.

In [King et al. 2006b], the authors give a sparse network implementation of their protocols from [King et al. 2006a]. It is easy to see that everywhere agreement is impossible in a sparse network where the number of faulty processors t is sufficient to surround a good processor. A protocol in which processors use $o(n)$ bits may seem as vulnerable to being isolated as in a sparse network, but the difference is that without access to private random bits, the adversary can't anticipate at the start of the protocol where communication will occur. In [Holtby et al. 2008], it is shown that even with private channels, if a processor must pre-specify the set of processors it is willing to listen to at the start of a round, where its choice in each round can depend on the outcome of its random coin tosses, at least one processor must send $\Omega(n^{1/3})$ messages to compute Byzantine agreement with probability at least $1/2 + 1/\log n$. Hence the only hope for a protocol where every processor sends $o(n^{1/3})$ messages is to design outside this constraint. We note that *a.e.BA* falls within this restrictive model, but *a.e.BA-to-BA* does not, as the decision of whether a message is listened to (or acted upon) depends on how many messages carrying a certain value are received so far.

3. ALMOST EVERYWHERE BYZANTINE AGREEMENT

We now outline our main algorithm, *a.e.BA*. The processors are arranged into nodes in a q-ary tree. Each processor appears in polylogarithmic places in each level of the tree, so that a large fraction of nodes on each level must contain a majority of good processors. The levels of the tree are numbered from the leaf nodes (level 1) to the root (level ℓ^*). Nodes at higher levels contain more processors; the root contains all processors.

At the start, each processor p_i, generates an *array* of random bits, consisting of one *block* for each level of the network and secret shares each block with the processors in the i^{th} node on level 1.

Beginning with the lowest level of the tree, (the processors of) each node runs an election among r arrays from which a subset of w arrays are selected. To run this election at level ℓ, the ℓ block of each array supplies a random bin choice and random bits to run *a.e.BA-with-random-source* to agree on each bin choice of every competing array. The shares of the remaining blocks of arrays which remain in the competition are further subdivided into more shares and split among the larger number of processors in the parent node (and erased from the current processors' memories.) In this way, as an array becomes more important, an adversary cannot learn its secret value by taking over the smaller number of processors on lower levels which used to share the secret.

Random bits are revealed as needed by sending the iterated shares of secrets down paths to *all* the leaves of the subtree rooted at the node where the election is occurring. In particular, at each level ℓ, ℓ-shares are collected to reconstruct

$(\ell - 1)$-shares. In the level 1 nodes, each processor sends the other processors its 1-share to reconstruct the original secret.

The winning arrays of a node's election compete in elections at the next higher level. At the root which contains all processors, there are a small number of arrays which can be used to run *a.e.BA* or *a.e.random-source*, to produce the output of the protocol.

The method of secret sharing and iterative secret sharing is described in Section 3.1. Networks and communication protocols are described in Section 3.2; the election routine is described in Section 3.4. The procedure for running *a.e.BA-with-random-source* is described in Section 3.3. The main procedure for *a.e.BA* is in Section 3.5. The extension of the almost everywhere Byzantine Agreement protocol to a solution for *a.e.random-source* is in Section 3.7. Finally the analysis and correctness proof can be found in Sections 3.8 and 3.9, respectively.

3.1 Secret sharing

We assume any secret sharing scheme which is a (n, τ) threshold scheme, for $\tau = n/3$. That is, each of n players are given shares of size proportional to the message M and τ shares are required to reconstruct M. Every message which is the size of M is consistent with any subset of fewer than τ shares, so no information as to the contents of M is gained about the secret from holding fewer than τ shares. Furthermore, we require that if a player possesses all the shares and less than $n/3$ are falsified by an adversary, the player can reconstruct the secret. See [McEliece and Sarwate 1981] for details on constructing such a scheme. We will make extensive use of the following definition.

Definition 1. *secretShare(s): To share a sequence of secret words s with n_1 processes (including itself) of which $\tau_1 - 1$ may be corrupt, a processor (dealer) creates and distributes shares of each of the words using a (n_1, τ_1) secret sharing mechanism. Note that if a processor knows a share of a secret, it can treat that share as a secret. To share that share with n_2 processors of which at most $\tau_2 - 1$ processors are corrupt, it creates and distributes shares of the share using a (n_2, τ_2) mechanism and deletes its original share from memory. This can be iterated many times. We define a 1-share of a secret to be a share of a secret and an i-share of a secret to be a share of an $(i-1)$-share of a secret.*

To reveal a secret sequence s, all processors which receive a share of s from a dealer send these shares to a processor p which computes the secret. This also may be iterated to first reconstruct $i-1$ shares from i shares, etc., and eventually the secret sequence.

Lemma 1. *If a secret is shared in this manner up to i iterations, then an adversary which possesses less than τ_1 1-shares of a secret and for $1 < j \leq i$, less than τ_j j-shares of each $(j-1)$-share that it does not possess, learns no information about the secret.*

Proof. The proof is by induction. For level 1, it is true by definition of secret sharing. Suppose it is true up to i iterations.

Let v be any value. By induction, it is consistent with the known $\tau_j - 1$ shares on all levels $j \leq i$ and some assignment S_i of values to sets of unknown $n_i - \tau_i + 1$

i-shares. Then consider the shares of these shares that have been spread to level $i+1$. For each value of an i-share given by S_i, there is an assignment S_{i+1} of values to the unknown $n_{i+1} - \tau_{i+1} + 1$ shares consistent with that value. Hence knowing in addition the $\tau_{i+1} - 1$ $(i+1)$-shares of each i-share does not reveal any information about the secret. $\square$

3.2 Network and Communication

We first describe the topology of the network and then the communications protoocols.

3.2.1 *Samplers.* Key to the construction of the network is the definition of an averaging sampler which was also used heavily in [Kapron et al. 2009; King et al. 2006b]. We repeat the definition here for convenience. Our protocols rely on the use of averaging (or oblivious) samplers to determine the assignment of processors to nodes on each level and to determine the communication links between processors. Samplers are families of bipartite graphs which define subsets of elements such that all but a small number contain at most a fraction of "bad" elements close to the fraction of bad elements of the entire set. Intuitively, they are used in our algorithm in order to generate samples that do not have too many bad processors. We assume either a nonuniform model in which each processor has a copy of the required samplers for a given input size, or else that each processor initializes by constructing the required samplers in exponential time.

DEFINITION 2. *Let $[r]$ denote the set of integers $\{1, \dots, r\}$, and $[s]^d$ the multisets of size d consisting of elements of $[s]$. Let $H : [r] \to [s]^d$ be a function assigning multisets of size d to integers. We define the size of the intersection of a multiset A and a set B to be the number of elements of A which are in B.*

Then we say H is a (θ, δ) sampler *if for every set $S \subset [s]$ at most a δ fraction of all inputs x have $\frac{|H(x) \cap S|}{d} > \frac{|S|}{s} + \theta$.*

The following lemma establishing the existence of samplers can be shown using the probabilistic method. For $s' \in [s]$, let $deg(s') = |r' \in [r] \mid s.t.\ s \in H(r')\}|$. A slight modification of Lemma 2 in [Kapron et al. 2009] yields:

LEMMA 2. *For every $r, s, d, \theta, \delta > 0$ such that $2\log_2(e) \cdot d\theta^2\delta > s/r + 1 - \delta$, there exists a (θ, δ) sampler $H : [r] \to [s]^d$ and for all $s \in [s]$, $deg(s) = O((rd/s)\log n)$.*

Note that limiting the degree of s limits the number of subsets that any one element appears in.

For this paper we will use the term *sampler* to refer to a $(1/\log n, 1/\log n)$ sampler, where $d = O((s/r+1)\log^3 n)$, unless otherwise specified.

3.2.2 *Network structure.* Let P be the set of all n processors. The network is structured as a complete q-ary tree. The level 1 nodes (leaves) contain $k_1 = \log^3 n$ processors. Each node at height $\ell > 1$ contains $k_\ell = q^\ell k_1$ processors; there are $(n/k_\ell)\log^3 n$ nodes on level ℓ; and the root node at height $\ell^* = \log_q(n/k_1)$ contains all the processors. The contents of each node on level ℓ is determined by a sampler where $[r]$ is the set of nodes, $[s] = P$ and $d = k_\ell$.

There are n leaves, each assigned to a different processor, assuming n is a power of q. If n is not a power of q, we will blow up the size of the problem by a factor between q and q^2 as follows: let a be the smallest power of q such that $q^a > nq$. Assume the leaves are numbered 1 through q^a. Assign processor i to all leaves numbered $i \; mod \; n$. For each leaf, the processor generates a new array. In this way, each processor is assigned to either $x = \lfloor q^a/n \rfloor$ or $x+1$ leaves, where $q^2 > x > q$. In the worst case, each bad processor is assigned to $x+1$ leaves and each good processor is assigned to x leaves. If there are $2/3+\epsilon$ fraction of good processors then there are more than $2/3+\epsilon-1/q$ fraction of leaves which have been assigned to good processors. As q will be set to a function which is polylogarithmic in n, this fraction remains $2/3+\epsilon'$ for a constant ϵ'. Note that changing the problem size by a polylogarithmic factor does not affect the complexity as stated in the main results. From now on, we will describe the algorithm assuming that n is a power of q.

The edges in the network are of three types:

(1) *Uplinks:* The *uplinks* from processors in a child node on level ℓ to processors in a parent node on level $\ell+1$ are determined by a sampler of degree $d = q\log^3 n$, $[r]$ is the set of processors in the child node and $[s]$ is the set of processors in the parent node. Let C, C' be child nodes of a node A. Then the mapping of processors in C, C' to $[r]$ (and A to $[s]$) determines a *correspondence* between the uplinks of C and C'.

(2) $\ell - links$: The $\ell - links$ between processors in a node C at any level $\ell > 1$ to C's descendants at level 1 are determined by a sampler where $[r]$ is the set of processors in the node C and $[s]$ is C's level 1 descendants. Here, $r = q^\ell k_1$; $s = q^{\ell-1}$; $d = O(\log^3 n)$ and the maximum number of $\ell - links$ incident to a level 1 node is $O(qk_1\log^4 n)$.

(3) *Links between processors in a node* are also determined by a sampler of polylogarithmic degree. These are described in *a.e.BA-with-random-source*.

Definition 3. *Call a node* good *if it contains at least a $2/3+\epsilon$ fraction of good processors. Call it* bad *otherwise.*

From the properties of samplers, we have:

(1) Less than a $1/\log n$ fraction of the nodes on any level are bad.

(2) Less than a $1/\log n$ fraction of processors in every node whose uplinks are connected to fewer than a $2/3+\epsilon-1/\log n$ fraction of good processors, unless the parent or child, resp. is a bad node. We call such a set of uplinks for a processor *bad*.

(3) If a level ℓ node C has less than a $1/2-\epsilon$ fraction of bad level 1 descendants, then less than a $1/\log n$ fraction of processors in C are connected through $\ell - links$ to a majority of bad nodes on level 1.

3.2.3 *Communication protocols.* We use the following three subroutines for communication. Initially each processor p_i shares its secret with all the processors in the i^{th} node at level 1.

sendSecretUp(s): To send up a sequence s of secret words, a processor in a node uses *secretShare*(w) to send to each of its neighbors in its parent node (those connected by *uplinks*) a share of each word w of s. Then the processor erases s from its own memory.

sendDown(w, i): After a secret w has been passed up a path to a node C, the secret can be recovered by passing it down to the processors in the 1-nodes in the subtree. To send a secret word w down the tree, each processor in a node C on level i sends its i-shares of w *down* the uplinks it came from plus the corresponding uplinks from each of its other children. The processors on level $i-1$ receiving the i-shares use these shares to reconstruct $(i-1)$-shares of w. This is repeated for lower levels until all the 1-shares are reconstructed by the processors in all the 1-nodes in C's subtree. The processors in the 1-node each send each other all their shares and reconstruct the secrets received. Note that a processor may have received an i-share generated from more than one $i-1$ share because of the overlapping of sets (of uplinks) in the sampler.

sendOpen(w, ℓ) : This procedure is used by a node C on any level ℓ to learn a word w held by the set of level 1 nodes in C's subtree. Each processor in each level 1 node A sends w up the $\ell - links$ from A to a subset of processors in C. A processor in C receiving a version of w from each of the processors in a level 1 node takes a majority to determine the node A's version of w. Then it takes a majority over the values obtained from each of the level 1 nodes it is linked to.

3.2.4 *Correctness of communications*

DEFINITION 4. *A good path up the tree is a path from leaf to root which has no nodes which become bad during the protocol.*

LEMMA 3. (*1*) *If sendSecretUp*(s) *is executed up a path in the tree and if the adversary learns a word of the secret s before sendDown*(w, ℓ) *is executed for any ℓ, there must be at least one bad node on that path.*

(*2*) *Assume that s is generated by a good processor and sendSecretUp*(s) *is executed up a good path in a tree to a node A on level ℓ, followed by sendDown*(w, ℓ) *where w is a word of s, and then sendOpen*(w)*. Further assume there are at least a $1/2 + \epsilon$ fraction of nodes among A's descendants on level 1 which are good, and whose paths to A are good. Then a $1 - 1/\log n$ fraction of the good processors in A learn w.*

PROOF. Proof of (1): If *sendSecretUp*(s) is executed up a good path of length ℓ, then all secrets passed up uplinks incident to a $2/3$ majority of good processors will remain secret, by Lemma 1.

We consider the effect of secrets passed up uplinks incident to less than $2/3$ majority of good processors. For the purpose of this proof, let us redefine a processor in a node as bad if it is controlled by the adversary, or it is good and its share is learned by the adversary. We show by induction on the level number that when secrets are passed from a leaf to level ℓ for any level ℓ, the level i node on a good path up to level ℓ contains no more than $1/3 - \epsilon + 1/\log n$ fraction of processors which are bad.

For the basis case, $i = \ell$, only the shares received by bad processors in the level ℓ node are learned by the adversary. Since the node is good, there are no more than a $1/3 - \epsilon$ fraction of bad processors.

We assume by induction that no more than $1/3 - \epsilon + 1/\log n$ processors in nodes on levels i through ℓ are bad. We show it is true for level $i-1$. Since the uplinks are determined by a $(1/\log n, 1/\log n)$-sampler, when the processors in a node on level $i-1$ pass the secret shares up the uplinks, no more than a $1/\log n$ fraction of the uplink sets are incident to more than $1/3 - \epsilon + 2/\log n$ fraction of bad processors on level i. The adversary through these processors may thus learn a $1/\log n$ fraction of $(i-1)$-shares sent by processors in the level $i-1$ node in the path. Thus no more than a $1/\log n$ fraction of processors holding the $(i-1)$-shares are bad in addition to the bad processors in the level $i-1$ node, for a total fraction of $1/3 - \epsilon - 1/\log n$ bad processors. This completes the induction.

We have shown that no more than a $1/3 - \epsilon + 1/\log n$ fraction of 1-shares will be learned by the adversary. Thus, the adversary cannot learn the secret, which completes the proof of part (1) of the lemma.

Proof of (2): The proof of part (1) shows that on each level of a good path, no more than a $1/\log n$ fraction of good processors in the path have uplinks which are incident to less than a $2/3$ fraction of good processors. Hence when the i-shares are returned down the uplinks they were sent, for all but $1/\log n$ processors, each receives enough shares to recover the $i-1$ share that it once had. The same is true for the processors in the other good paths of the subtree rooted at A. In particular, all but a $1/\log n$ fraction of processors learn the secret in each of the leaves of the subtree rooted at A, for all leaves on good paths to A. If there are at least a $1/2+\epsilon$ fraction of such leaves then by property (3) of the samplers, at least a $1 - 1/\log n$ fraction of the processors in A will be connected by ℓ-links to a majority of such leaves. Thus when *sendOpen* is executed, a $1 - 1/\log n$ fraction of processors in A will learn the secret correctly when they take the majority of values held by each leaf to which they are connected by ℓ-links, where that value is determined by a majority of processors in the leaf. □

3.3 a.e.BA with random sequence

In this section, we present the *a.e.BA-with-random-source* algorithm and the proof of its correctness (Theorem 4). We assume this is run on a subset of size k of the n processors where the fraction of bad processors in the subset is no more than $1/3-\epsilon$ for some fixed $\epsilon > 0$. We assume access to a sequential (s, t) *a.e.random-source*, where s and t are polylogarithmic, which outputs sequences of bits, one per call.

The algorithm is an implementation of Rabin's global coin toss Byzantine agreement protocol except that the coin toss is a.e., rather than global. In addition, only some of the coin tosses are random, and broadcast is replaced by sampling a fraction of the other processors according to the edges of a certain type of sparse, directed mulitgraph. This graph has the property that almost all of the samples contain a majority of processors whose value matches the value of the majority of the whole set. In the algorithm, the value ϵ_0 is set to any quantity less than $(3/4)\epsilon$ (e.g. $\epsilon/2$), as will be shown in the analysis.

THEOREM 4. *[a.e.BA-with-random-source] Given a set of processors such that a $2/3+\epsilon$ fraction are good and given a (s,t) a.e. random sequence of s bits, then for any fixed, positive integer d,* a.e.BA-with-random-source *runs in time $O(s)$ with bit complexity $O(s \log^d n)$ per processor. With probability at least $1 - 1/2^{t-1}$, all but*

Algorithm 1 *a.e.BA-with-random-source*

```
Set $vote \leftarrow b_i$; For $s$ rounds do the following:
(1) Send $vote$ to all its successors in $G$;
(2) Collect votes from its predecessors in $G$;
(3) $maj \leftarrow$ majority bit among votes received;
(4) $fraction \leftarrow$ fraction of votes received for $maj$;
(5) $coin \leftarrow$ result of call to a.e.random-source;
(6) If $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$ then $vote \leftarrow maj$
(7) else
    (a) If $coin$ = "heads", then $vote \leftarrow 1$, else $vote \leftarrow 0$;
At the end of all rounds, commit to $vote$ as the output bit;
```

a $1/\log^d n$ fraction of the good processors commit to the same vote b. Further, if all but a $1/\log n$ fraction of good processors have the same input bit b then b is the output bit.

We'll need an easy corollary which is the same except the goal is to agree on a word rather than a bit and s is a sequence of words.

Corollary 1. *Given a set of processors such that a $2/3+\epsilon$ fraction are good and given a (s,t)* a.e.random-source *which generates s words of size w, there is an algorithm which runs in time $O(s)$ with bit complexity $O(sw\log^d n)$ per processor, such that with probability at least $1-w/2^{t-1}$, all but a $w/\log^d n$ fraction of the good processors output the same word. Moreover, if all but a $1/\log n$ fraction of good processors have the same input word b then b is the word output by all but a $w/\log^d n$ fraction of the good processors.*

Proof. We can regard s as w separate streams. By a union bound, the probability that any bit of the word is not decided correctly is less than the sum of the probabilities of failure or $w/2^{t-1}$. For each bit of b, fewer than $1/\log^d$ fraction of good processors are not in agreement, for a total of less than $w/\log^d n$ fraction of good processors. □

Before proving Theorem 4 we establish the following lemmas.

For a fixed round, let $b' \in \{0,1\}$ be the bit that the majority of good processors vote for in that round. Let S' be the set of good processors that will vote for b' and let $f' = |S'|/k$. Let $0 \leq \epsilon_0 \leq$ be a fixed constant to be determined later. We call a processor *informed* for the round if the fraction value for that processor obeys the following inequalities:

$$(1-\epsilon_0)f' \leq fraction \leq (1+\epsilon_0)(f'+1/3-\epsilon)$$

The following lemma establishes the existence of sparse graphs that ensure all but a $1/\log^d n$ fraction of the good processors are informed in every round of the algorithm. Note that it is straightforward to show that a graph with the desired properties can be constructed with probability $1-1/n^c$ for any fixed c in polynomial time.

Lemma 4 Graph existence. *For any positive k and positive constants ϵ_0 and d, there exists a directed multigraph G on k vertices with maximum out-degree no* *more than $C_3 \log^d n$, where C_3 depends only on ϵ_0 and d, such that if this communication graph is used in Algorithm 1, then in every round all but a $1/\log^d n$ fraction of the good processors are informed.*

PROOF. Let G be a multigraph with k nodes which is constructed as follows: each node chooses $C_3 \log n$ neighbors uniformly at random with replacement. Fix the set S'. We know that S' is of size at least $(1/3 + \epsilon/2)k$ since at least half of the good processors must vote for the majority bit. Let $f' = |S'|/k$. We will also fix a set T which consists of all the processors that have $fraction < (1 - \epsilon_0)f'$. We will first show that the probability that T is of size $k/2 \log^d n$ is very small for fixed S' and T, and will then show, via a union bound, that with high probability, for any S' there is no set of $k/2 \log^d n$ processors with $fraction < (1 - \epsilon_0)f'$. Finally, we will use a similar technique to show that with high probability, no more than $k/2 \log^d n$ processors have $fraction > (1+\epsilon_0)(f' + 1/3 - \epsilon)$. This will complete the proof.

To begin, we fix the set S' of size at least $(1/3+\epsilon/2)k$ and fix T of size $k/(2 \log^d n)$. Let $\xi(S', T)$ be the event that all processors in T have $fraction < (1 - \epsilon_0)f'$. We let X be the number of edges to nodes in T from nodes in S' and let Y be the number of total edges into T. If every node in T has $fraction < (1 - \epsilon_0)f'$, then $X/Y < (1 - \epsilon_0)f'$. We now bound the probability of that event. Let $\epsilon_1 = \frac{\epsilon_0}{2-\epsilon_0}$, Note that $E[X] = |S'|C_3 \log^d n|T|/k = f'kC_3/2$, and $E[Y] = kC_3/2$. Each edge out of S' falls into T independently with probability $|T|/k$. Thus we can apply Chernoff bounds to say that

$$Pr(X < (1 - \epsilon_1)E[X]) < e^{-E[X]\epsilon_1^2/2} = e^{-f'kC_3\epsilon_1^2/4}.$$

Similarly, we can use Chernoff bounds to say that

$$Pr(Y > (1 + \epsilon_1)E[Y]) < e^{-E[Y]\epsilon_1^2/3} = e^{-C_3k\epsilon_1^2/6}.$$

Hence the probability that either of these bounds is violated is no greater than the sum of these two probabilities, or less than $1/e^{Ck}$ for any constant C and $C_3 = 6C/\epsilon_1{}^2$. Thus, $Pr(X/Y < \frac{(1-\epsilon_1)f'}{1+\epsilon_1}) < e^{-Ck}$. Substituting for ϵ_1 we have that $Pr(X/Y < (1 - \epsilon_0)f')) < e^{-Ck}$.

Let ξ be the union of events $\xi(S', T)$ for all possible values of S' and T. Then we know by union bounds that

$$\begin{aligned} Pr(\xi) &= \sum_{S',T} Pr(\xi(S', T)) \\ &\leq 2^k 2^k e^{-Ck} \\ &< 1/2 \end{aligned}$$

Where the last equation holds provided that C is sufficiently large, but depending only on ϵ_0. We have thus shown that with probability greater than 1/2, the number of processors with $fraction < (1 - \epsilon_0)f'$ is no more than $k/(2 \log^d n)$.

We note that the set of processors violating the upper bound is maximized when all bad processors vote for the majority bit. By a similar analysis, letting S'' consist of the union of S' and the set of bad processors, we can show that with high probability, the number of processors with $fraction > (1+\epsilon_0)(f' + 1/3 - \epsilon)$ is no more than $k/(2\log^d n)$. The proof is straightforward, but is included here for completeness. Let T now consist of all processors that have $fraction > (1 + \epsilon_0)(f' + 1/3 - \epsilon)$. Let $f'' = f' + 1/3 - \epsilon$. For fixed sets S'' and T, let $\xi'(S'', T)$ be the event that all processors in T have $fraction > (1+\epsilon_0)f''$.

We let X be the number of edges to nodes in T from nodes in S'' and let Y be the number of total edges into T. If every node in T has $fraction > (1+\epsilon_0)f''$, then $X/Y > (1+\epsilon_0)f''$. We now bound the probability of that event. Let $\epsilon_1 = \frac{\epsilon_0}{2+\epsilon_0}$ Note that $E[X] = |S''|C_3 \log^d n|T|/k = f''kC_3/2$ and $E[Y] = C_3k/2$. Each edge out of S'' falls into T independently with probability $|T|/k$. Thus we can apply Chernoff bounds to say that

$$Pr(X > (1+\epsilon_1)E[X]) < e^{-E[X]\epsilon_1^2/3} = e^{-f''kC_3/6}.$$

Similarly, we can use Chernoff bounds to say that

$$Pr(Y < (1-\epsilon_1)E[Y]) < e^{-E[Y]\epsilon_1^2/2} = e^{-C_3k\epsilon_1^2/4}.$$

Hence the probability that either of these bounds is violated is no greater than the sum of these two probabilities, or less than $1/e^{Ck}$ for any constant C and $C_3 = 4C/\epsilon_1^2$. Thus, $Pr(X/Y > \frac{(1+\epsilon_1)f''}{1-\epsilon_1}) < e^{-Ck}$. Substituting for ϵ_1 we have that $Pr(X/Y > (1+\epsilon_0)f")) < e^{-Ck}$.

Let ξ be the union of events $\xi(S'', T)$ for all possible values of S'' and T. Then we know by union bounds that

$$\begin{aligned} Pr(\xi) &= \sum_{S'',T} Pr(\xi(S'', T)) \\ &\leq 2^k 2^k e^{-Ck} \\ &< 1/2 \end{aligned}$$

Where the last equation holds provided that C is sufficiently large, but depending only on ϵ_0. We have thus shown that with probability greater than $1/2$, the number of processors with $fraction > (1+\epsilon_0)f$" is no more than $k/(2\log^d n)$.

These two results together establish that the existence of a graph with k nodes and outdegree $\min\{k, C_3 \log^d n\}$ such that in any round of the algorithm, all but a $1/\log^d n$ fraction of processors are informed provided that C_3 is chosen sufficiently large with respect to ϵ_0. $\square$

We can now show the following lemma.

Lemma 5. *Let n be sufficiently large compared to ϵ. Then, in any given round if all but a $2/\log n$ fraction of good processors vote for the same value b', then for every remaining round, all but a $1/\log n$ fraction of good processors will vote for b'.*

Proof. We will show that if all but a $2/\log n$ fraction of good processors vote for the same value b' in some round i, then in round $i+1$, all but a $1/\log n$ fraction of good processors will vote for b'; the result then follows by induction. Consider what happens after the votes are received in round i. We know that for this round, $f' \geq 2/3 + \epsilon - 2/\log n \geq 2/3 + \epsilon/2$ for n sufficiently large. Thus, every informed processor in round i will have $fraction \geq (1-\epsilon_0)f' \geq (1-\epsilon_0)(2/3+\epsilon/2)$; and so every informed processor will set its vote value, at the end of the round, to b'. It follows that all the processors that were informed in round i will vote for b' in round $i+1$. By Lemma 4, we can ensure that in every round, all but a $1/\log n$ fraction of processors are informed. Note that this result holds irrespective of the outcome of *a.e.random-source* for the given round and successive rounds, even including the case where different processors receive different outcomes from that subroutine. □

LEMMA 6. *If the call to* a.e.random-source *succeeds in some round, i.e. all but a* $1/\log n$ *fraction of good processors receive an unbiased random bit, then with probability at least* $1/2$*, at the end of that round, all but a* $2/\log n$ *fraction of good processors will have a vote value equal to the same bit.*

PROOF. Fix a round where the call to *a.e.random-source* succeeds. There are two main cases

Case 1: No informed processor has $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$. In this case, at the end of the round, all informed processors who receive the *a.e.random-source* random bit will set their vote to the same bit. By Lemma 4, this means that all but $2/\log n$ processors will set their vote to the same bit.

Case 2: At least one informed processor has $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$. We first show that in this case, all informed processors that have $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$ will set their vote to the same value at the end of the round. We show this by contradiction. Assume there are two processors, x and y, where $fraction_x$ ($fraction_y$) are the fraction values of x (y, resp.), such that both $fraction_x$ and $fraction_y$ are greater than or equal to $(1-\epsilon_0)(2/3+\epsilon/2)$, and x sets its vote to 0 at the end of the round, while y sets its vote to 1.

Let f'_0 (f'_1) be the fraction of good processors that vote for 0 (1) during the round. Then we have that $fraction_x \geq (1-\epsilon_0)(2/3+\epsilon/2)$. By the definition of informed, we also know that $fraction_x \leq (1+\epsilon_0)(f'_0+1/3-\epsilon)$. This implies that

$$(1-\epsilon_0)(2/3+\epsilon/2) \leq (1+\epsilon_0)(f'_0+1/3-\epsilon).$$

Isolating f'_0 in this inequality, we get that

$$f'_0 \geq \frac{1/3+(3/2)\epsilon-\epsilon_0+(1/2)\epsilon\epsilon_0}{1+\epsilon_0}.$$

A similar analysis for $fraction_y$ implies that

$$f'_1 \geq \frac{1/3+(3/2)\epsilon-\epsilon_0+(1/2)\epsilon\epsilon_0}{1+\epsilon_0}.$$

But then we have that,

$$\begin{aligned} f'_0+f'_1 &\geq \frac{2/3+3\epsilon-2\epsilon_0+\epsilon\epsilon_0}{1+\epsilon_0} \\ &> 2/3+\epsilon \end{aligned}$$

where the last line is clearly a contradiction that holds provided that $\epsilon_0 < (3/4)\epsilon$.

Now, let b' be the value that all good and informed processors with $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$ set their value to at the end of the round. With probability

1/2, the outcome of *a.e.random-source* is equal to b' and in this case, all informed processors that receive the *a.e.random-source* random bit will set their vote to the same bit, i.e. all but $2k/\log n$ good processors. □

We can now prove Theorem 4.

PROOF. Lemma 5 establishes that if all but $1/\log n$ processors initially start with the same input bit, then all but a $1/\log n$ fraction of the good processors will eventually commit to that bit. Lemmas 6 and 5 together establish that the probability of having a round in which all but a $2/\log n$ fraction of good processors come to agreement, and after which all but a $1/\log n$ fraction of good processors will stay in agreement, is at least $1-2^{t-1}$ where t is the number of rounds in which *a.e.random-source* succeeds. A simple union bound on the probabilities of error implies the theorem. □

3.4 Arrays and the election subroutine

Here we describe the array of random bits generated by each processor. Each processor generates a sequence of blocks, each except the last to be used for the elections in the nodes along the path to the root. The last block is only used to decide the final output bit, and contains only the one bit needed to compute *a.e.BA-with-random-source*. To understand the blocks, we first describe Feige's election procedure, which is designed for an atomic broadcast model, and present a lemma about its performance.

Feige's election procedure*[Feige 1999]:*
Each candidate announces a bin number in some small range. The good candidates choose a bin number selected uniformly at random in this range. The winners are the candidates which choose the bin picked by the fewest.

LEMMA 7. *[Feige 1999] Given a set of r numbers in the range $[1,..,numBin]$, let S be subset of these numbers which are generated independently at random by good processors, where the remaining numbers are chosen by an adversary which can view S before deciding on its values. For any constant c and sufficiently large n, if $|S|/numBin \geq (2c+2)\log^3 n$, then with probability at least $1-1/n^c$, the fraction of winners which are from S is at least $|S|/r - 1/\log n$.*

PROOF. The expected number of winners from S in any bin is $|S|/numBin$. The probability that the fraction of winners deviates from its expectation for one bin by more than $1/\log n$ is $1 - e^{-|S|/(2numBin\log^2 n)} = 1 - 1/n^{c+1}$, using a Chernoff bound. The probability that this occurs for any bin, including the bin chosen by the fewest candidates, is $1-1/n^c$, by a union bound. □

We show how to adapt Feige's election method to a set of processors which communicate by message-passsing.

The *input* to an election is an a.e. sequential (r,s) random sequence of blocks labelled $B_1,...,B_r$. Each *block* B is a sequence of bits, beginning with an initial word (bin choice) $B(0)$ followed by r words $B(1),B(2),..,B(r)$. The number of bins is *numBins* and each word has $\log(numBins)$ bits. After the election, the *output* of each processor is a set of winning indices W. Let $w = |W|$. Given $r \geq 2\log^4 n$, we set $numBins$ so that $w = r/numBins = \log^4 n$.

We now describe the election subroutine.

(1) In parallel, for $i = 1, ..., r$, the processors run *a.e.BA-with-random-source* on the bin choice of each of the r candidate blocks. Round j of *a.e.BA-with-random-source* to determine i's bin choice is run using the i^{th} word of the j^{th} processor's block $B_j(i)$.
Let $b_1, ..., b_r$ be the decided bin choices.

(2) Let $min = i$ s.t. $|\{j \mid b_j = i\}|$ is minimal.
$W \leftarrow \{j \mid b_j = min\}$.
(If $|W| < r/numBins$ then W is augmented by adding the first $r/numBins - |W|$ indices that would otherwise be omitted.)

Then Feige's result can be extended to this context as follows:

LEMMA 8. *In the election subroutine, let the number of candidate blocks be $r \geq 2\log^4 n$. Let S be the subset of candidate blocks generated independently at random by good processors and assume $|S| \geq (2c+2)r/\log n$. If the set of processors running the election subroutine has a $2/3+\epsilon$ fraction of good processors, then with probability $1-1/n^c$ the election results in agreement on a set of winners such that the fraction of winners from S is at least $|S|/r - 1/\log n$ and the winners are agreed to by a $1-1/\log n$ fraction of good processors. Choosing d so that $r \log numBins < \log^d n$, the election algorithm runs in time $O(r)$ and $O(r \log^d n)$ bits per processor.*

PROOF. Consider the r bin choices together as one word of $r \log numBins$ bits. Then let St consist of the sequence of blocks B_i without their first words containing the bin choices. Let $s = |S|$. Then St is an a.e. (r, s) random sequence of words of length $r \log numBins$. By Corollary 1 all but a $r \log numBins/\log^d n < 1/\log n$ fraction of good processors agree on the bin choices and hence the winners, with probability $1 - r \log numBins/2^s > 1 - 1/n^{c+1}$ for any constant c. Since $numBins$ was chosen so that $r/numBins = \log^4 n$, it follows that $s/numBins \geq (2c + 2)r/(numBins \log n) \geq (2c+2)\log^3 n$. Therefore, by Lemma 7, the fraction of winners that are good is $|S|/r - 1/\log n$ with probability $1 - 1/n^{c+1}$ for any constant c. Taking the union bound, both conditions hold w.h.p. The time and bit complexity follow from Corollary 1. □

3.5 Main protocol for a.e. BA

The main protocol for *a.e.BA* is given as Algorithm 2. Figure 1, which we now describe, outlines the main ideas behind the algorithm. The left part of Figure 1 illustrates the algorithm when run on a 3-ary network tree. The processors are represented with the numbers 1 through 9 and the ovals represent the nodes of the network, where a link between a pair of nodes illustrates a parent-child relationship. The numbers in the bottom part of each node are the processors contained in that nodes. Note that the size of these sets increase as we go up the tree. Further note that each processor is contained in more than one node at a given level. The numbers in the top part of each node represent the processors whose arrays are candidates at that node. Note that the size of this set (3) remains constant as we go from level 2 to level 3. Further note that the array of each processor is a candidate in at most one node at a given level.

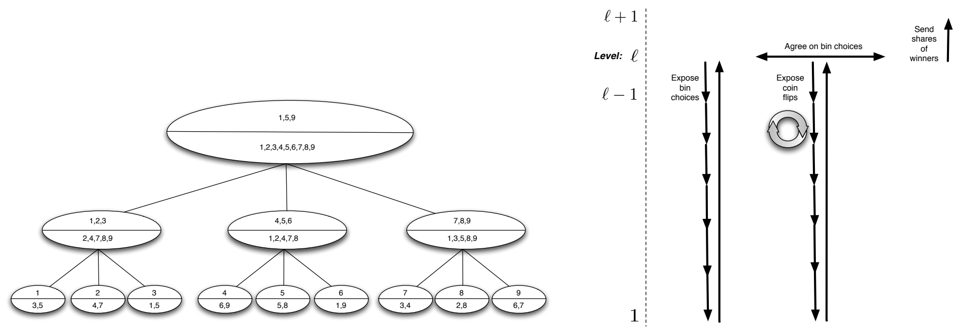

Fig. 1. Left: Example run of Algorithm 2 *a.e.BA* on a small tree; Right: Communication in different phases of Algorithm 2 for a fixed level ℓ.

The right part of Figure 1 illustrates communication in *a.e.BA* for the election subroutine occurring at a fixed node at level ℓ. This figure illustrates three main parts of the election subroutine algorithm (time moves from left to right). First, on the left, is "Expose bin choices". Here, the bin choices of the candidates at the level ℓ node are revealed in two parts: (1) hop-by-hop communication down the network using the *sendDown* protocol from level ℓ to level 1, at the end of which the nodes at level 1 know the relevant bin choices; and (2) direct communication up the network using the *sendOpen* protocol of these bin choices from the level 1 nodes to the level ℓ node.

The second part of the election subroutine is "Agree on bin choices". This involves running *a.e.BA-with-random-source* which uses "horizontal" communication among the processors in the node at level ℓ that are running *a.e.BA-with-random-source*. It requires the output of an *a.e.random-source*. This is provided by communication within the subtree rooted at the node via hop-by-hop communication down the network, using *sendDown*, at the end of which the nodes at level 1 know the outcome of the next coin toss; and direct communication up the network, using *sendOpen*, of the coin toss outcome from the level 1 nodes to the level ℓ node.

The third and final part of the election subroutine is "Send shares of winners", where shares of the arrays of the winners at the level ℓ node are sent to the level $\ell+1$ parent node, via use of the procedure *sendSecretUp*.

3.6 Specifics of Algorithm 2

We set $q = \log^{\delta} n$ for $\delta > 4$ a constant. Recall that $w = \log^4 n$; r denotes the number of candidates in an election; and the number of bins in the election subroutine is $numBins$ which is chosen so that $r/numBins = w$.

In Step 1, each processor i generates an array of $\ell^* - 1$ blocks of random bits. Each block except the last contain $1 + r$ words of length $\log numBins$. Each processor is assigned one of the n nodes on level 1. The processor secret shares each of its blocks individually with the processors in that level 1 node.

In Step 2, in an election in a level 2 node C, there is one candidate per child, or a total of $r = q$ candidates. At elections on higher levels there are $r = wq$ candidates since there are w candidates which won in each child node. If an array wins every election it participates in, up to level $\ell < \ell^*$, then its $\ell - 1$ block is used as input to to decide the election at the level ℓ ancestor of its assigned level 1 node. The first word of the block is its bin choice. The r remaining words in the block are used as a random source for one round of the *a.e.BA-with-random-source* to concurrently decide all the bits of all bin choices of all the r candidates in the election. Each random word required by each of the r rounds of the *a.e.BA-with-random-source* is provided by a different candidate array.

As mentioned above, *a.e.BA-with-random-source* is carried out using "horizontal communication," links between processors in the same node C. These links are determined by a sampler of degree $\log^d n$ where d is chosen so that $\log^d n > r \log numBins$ as described in Lemma 8. Note that d is a constant which depends linearly on the δ in Theorem 2.

In Step 3, *a.e.BA-with-random-source* is carried out by all the processors (which are all in the root node) to decide on one of their input bits, using as a random source the sequence of single bits provided by the last blocks of the $r = wq$ winners from the previous level. These are exposed in turn, using *sendDown* and *sendOpen* as described above.

3.7 Modification to a.e. random-sequence

The protocol *a.e.BA* can be modified easily to solve $(s, 2s/3)$-*a.e.random-source* for s a sequence of wq words. Add one more block of the desired length to each processor's array at the start. At level ℓ^*, use sendDown and sendOpen to recover each word, one from each of the wq contestants. The time and bit complexity is given in Theorem 2.

3.8 Bit complexity and running time analysis

We now show:

LEMMA 9. *For any $\delta > 0$, the Almost Everywhere Byzantine Agreement protocol requires $\tilde{O}(n^{4/\delta})$ bits per processor and runs in time $O((\log n)^{6+\delta}/(\log\log n)^2)$.*

PROOF. We first analyze running time. The running times of Step 1 and Step 2(c) is $O(1)$. For an election on level ℓ, Step 2(a) requires $O(\ell)$ time to execute *sendDown*. Step 2(b) requires r rounds, each dominated by the cost of *sendDown* for a total running time of $O(r\ell)$. There are $r = q$ rounds in the first execution of Step 2(b) and $r = wq$ rounds on the second and later executions of Step 2(b) and in Step 3. Each round *a.e.BA-with-random-source* takes $O(1)$ time plus the time needed to execute *sendDown* which is proportional to the level of the node running the election. Thus, the total running time is $O(\sum_{\ell=2}^{\ell^*} \ell wq) = O((\ell^*)^2 wq) = O((\log n/\log\log n)^2 \log^4 n \log^\delta n) = O(\log^{6+\delta}/(\log\log n)^2)$.

We now consider the number of bits communicated per processor. The assignment of processors to nodes on a given level is determined by a sampler. By Lemma 2, no processor is assigned to more than $deg(s) = O(\log^4 n)$ nodes in a level, or $O(\ell^* \log^4 n)$ over all levels. Hence it suffices to bound the cost per appearance of a processor in a node to get a $\tilde{O}$ result. Step 1 requires each processor to generate less than $\ell^* wq$ random words of $\log numBins = O(\log\log n)$ number of bits. Each share takes the same number of bits as the shared secret, and there are k_1 shares. When a processor in a level 1 node receives its share, it shares it with its parent

Algorithm 2 *a.e.BA*

```
(1) Generate Secrets and Send Up Shares:
    For all $i$ in parallel
    (a) Each processor $p_i$ generates an array of $\ell^* - 1$ blocks $B_i$ and uses
        $secretshare$ to share its array with the $i^{th}$ level 1 node;
    (b) Each processor in the $i^{th}$ level 1 node uses $sendSecretUp$ to share its
        1-share of $B_i$ with its parent node and then erases its shares from memory.

(2) Elect Winners at Root Node:
    Repeat for $\ell = 2$ to $\ell^* - 1$
    (a) Expose bin choices:
        For each processor in each node $C$ on level $\ell$:
        (Define the following: For $t = 1, ..., w$ and $i = 1, ..., q$, let $B_{(i-1)w+t}$ be the
        $t^{th}$ array sent up from child $i$. If $\ell = 2$ then $w$ is replaced by 1 and $r = q$;
        else $r = wq$
        $W \leftarrow B_1 \| B_2 \| ... \| B_r$
        Let $F$ be the sequence of first blocks of the arrays of $W$, i.e., the $i^{th}$ array
        of $F$ is the first block of the $i^{th}$ array of $W$. Let $S$ be the sequence of the
        remaining blocks of each array of $W$.)
        In parallel, for all candidates $i = 1, 2, .., r$
        i.  $sendDown(F_i(1))$;
        ii. $sendOpen(F_i(1), \ell)$.
    (b) Agree on bin choices:
        If $\ell < \ell^*$ then for rounds $i = 1, ..., r$
        i.  Expose coin flips: Generate $r$ random words for the $i^{th}$ round of
            $a.e.BA$-$with$-$random$-$source$ to decide each of $r$ bin choices.
            In parallel, for all contestants $j = 1, .., r$
            A. $sendDown(F_i(j))$; upon receiving all 1-shares, level 1 processors
               compute the secret bits $F_i(j))$;
            B. $sendOpen(F_i(j), \ell)$.
        ii. Run the $i^{th}$ round of $a.e.BA$-$with$-$random$-$source$ in parallel to decide
            the bin choice of all contestants.
    (c) Send Shares of Winners Up to Next Level: (Define the following:
        Let $W'$ be the winners of the election decided from the previous step i.e.
        the lightest bin. Let $S'$ be the subsequence of $S$ from $W'$.) All processors
        in a node at level $\ell$ use $sendSecretUp(S')$ to send $S'$ to its parent node
        and erase $S'$ from memory.

(3) Root Node Runs BA using Arrays of Winners:
    All processes in the single node on level $\ell^*$ run $a.e.BA$-$with$-$random$-$source$ once
    using their initial inputs as inputs to the protocol (instead of bin choices) and
    the remaining block (of one bit) of each candidate.
    For rounds $i = 1, 2, ..., qw$ of $a.e.BA$-$with$-$random$-$source$,
    (a) $sendDown(F_i(1), i)$;
    (b) $sendOpen(F_i(1), \ell)$.
    (c) Use $F_i(1)$ as the coin for this round of $a.e.BA$-$with$-$random$-$source$.
```

node via $q \log^3 n$ uplinks, for a total of $O((q \log^3 n + k_1)(\ell^* wq))$ words sent by each processor.

We next consider Step 2(c). Let d_m be the maximum number of uplinks incident to a processor in C which are incident to processors in a child node. By Lemma 2, $d_m = O(\log^4 n)$. We show by induction on ℓ that the number of ℓ-shares received by a processor in a level ℓ node which are shares of any one candidate is $O(d_m^{\ell-1})$. A processor in a level 1 node receives one 1-share of the array for the processor assigned to that node. A processor on level 2 is incident to d_m uplinks from processors from any child node, and so receives no more than d_m 2-shares of the arrays passed up from that child. A processor C on level ℓ is incident to d_m uplinks from processors in its child. By induction, each processor in a level $\ell - 1$ node has received $O(d_m^{\ell-2})$ $\ell - 1$-shares of any one candidate, so each of d_m processors in a child will pass up $O(d_m^{\ell-2})$ ℓ-shares for any one candidate, for a total of $d_m^{\ell-1}$ ℓ-shares for any one candidate. There are no more than w candidates whose shares are passed up by a child node. The size of each share is proportional to the size of the secret and the secrets consist of $O(\ell^* wq)$ words. Hence, the total cost of Step 2(c) for a processor in a level ℓ node which is holding an election is $O(\ell^* d_m^{\ell-1} w^2 q)$ words.

A processor in a node C at level ℓ receives $O(\ell^* d_m^{\ell-1} w^2 q)$ words from each of its q children. Step 2(a) and 2(b) require q copies of these words, which were sent up, to be sent down the tree for each level. These copies are for the corresponding uplinks in every child, which adds a factor of q^2, for a total of $O(\ell^* d_m^{\ell^*} * (w^2 q^3)$ words sent down. At the level 1 nodes, all $k_1 = O(\log^3 n)$ processors exchange 1-shares of the $(wq)^2$ words used for the election, for a total cost of $O(k_1 (wq)^2)$ bits and then send these values up its $O(qk_1 \log^4 n)$ ℓ-links to C. In addition, by Lemma 8, each execution of *a.e.BA-with-random-source* requires $O(r \log^d)$ bits per processor where d is a constant which is $O(\delta)$. Since $r \leq wq$, the total cost for an election in any node for Step 2 is the cost of $O(\ell^* d_m^{\ell^*}(w^2 q^3 + wq \log^d + (k_1)^2 (w^2 q^3 \log^4 n) = \tilde{O}(d_m^{\ell^*}(w^2 q^3 + w^2 q^3) = \tilde{O}(d_m^{\ell^*})$ words.

The cost of Step 3 is dominated by the cost of Step 2, as it requires a single execution of *a.e.BA-with-random-source* with a sequence of bits rather than words.

Hence the total cost per processor is $\tilde{O}(d_m^{\ell^*})$. Since $\ell^* = \log(n/k_1)/\log q$, this is $\tilde{O}((c \log^4 n)^{\log n/k_1/\log q}) = \tilde{O}(2^{(\log c' + 4 \log\log n)(\log(n/k_1)/\log q}$. Substituting $k_1 = O(\log^3 n)$ and $q = (\log^\delta n)$, this is $\tilde{O}(n^{4/\delta})$ for $\delta > 4$.

The total running time is $O((\ell^*) wq)$. Substituting $\ell^* = \log(n/k_1)/\log q$, $w = \log^4 n$, and $q = \log^\delta n$, we have $O(\log^{6+\delta} n/(\log\log n)^2)$. $\square$

3.9 Proof of correctness

Here we give some definitions. The first two are repeated from Section 3.2.

—A *good node* is a node which contains $2/3 + \epsilon - 1/\log n$ fraction of good processors.

—A *good path* to a node A is a path of good nodes from a leaf node up the tree to node A.

—An *array* is *good on level* ℓ: for $\ell = 1$, if it was generated by a good processor which was assigned to a good leaf node; for $\ell > 1$, if it was good on level $\ell - 1$ and the winner of a good election on level $\ell - 1$.

—An *election at node A* is *good* if $2/3 + \epsilon$ processors in A are good and all but at most a $1/\log n$ fraction of good processors agree on a set of w winners which are representative, i.e., if f fraction of candidate arrays are good on level $\ell - 1$ then at least $f - 1/\log n$ fraction of the winning arrays are good on level ℓ.

LEMMA 10. *An election at node A on level ℓ is good w.h.p. if the following conditions hold*

(1) A is a good node;

(2) The number of arrays that are used as input to the election is $r \geq 2\log^4 n$ and the number of these which are good is at least $(2c+2)r/\log n$; and

(3) There is a $1/2+\epsilon$ fraction of level 1 nodes in A's subtree which have good paths to A.

PROOF. We claim the prefix of the arrays which are used as input to the election form an a.e. (r, s) random sequence, where $s \geq (2c+2)r/\log n$. From Condition (2), this fraction of arrays is good which by definition implies they have been generated by good processors uniformly at random. Also by definition of a good array, they have been passed up good paths. By Lemma 3(1), each word of each array which is passed up a good path will be unknown to the adversary before execution of *sendDown* for that word. Therefore, the adversary has no knowledge of words in good arrays which come later in the sequence when the values of words which come earlier are exposed and decided upon by the good processors. Condition (3) and the fact that good arrays are passed up good paths show that the pre-conditions of Lemma 3(2) are satisfied, which implies that the exposed words from good arrays will be known a.e. in A.

Condition (1) ensures that there are sufficiently many good processors to run *a.e.BA-with-random-source* to agree on the bin choices. Condition (2) ensures that w.h.p., the set of winners of the election will be representative by Lemma 8. □

We now lower bound the fraction of arrays that remain that are good.

LEMMA 11. *In the* a.e.BA *algorithm, w.h.p., at least a $2/3 - 5\ell/\log n$ fraction of winning arrays are good on every level ℓ. In particular, w.h.p., the protocol can generate a sequence of random words, of length $r = wq$ (one from each array at the root of the tournament tree) of which a $2/3 + \epsilon - 5/\log\log n$ fraction are random and known to $1 - 1/\log n$ fraction of good processors.*

PROOF. With high probability, each good election causes an increase of no more than $1/\log n$ in the fraction of arrays that are bad. We now consider the number of good arrays discarded because of bad elections.

A violation of Condition (1) (the election node is bad) affects at most a $1/\log n$ fraction of arrays per level since at most a $1/\log n$ fraction of nodes are bad per level.

A violation of Condition (2) ($< (2c+2)r/\log n$ good arrays participating) cannot happen too often. As each array participates in one election on a level, an upper bound is a $(2c+2)/\log n$ fraction of the total number of arrays. As long as the fraction of bad arrays on a level is less than $1/3$, this is a $(2c+2)2/(3\log n)$ fraction.

A violation of Condition (3) ($< 1/2+\epsilon$ fraction of level 1 nodes have good paths to the current node) also cannot happen too often. In particular, a $1/\log n$ fraction of bad nodes can be responsible for making a $2/\log n$ fraction of elections bad in every level at above it , by making bad half the paths of those elections, thus eliminating an additional $2/\log n$ fraction of arrays that are good. Note that a bad election at a good node does not make additional paths bad, as information and secrets can still be passed through a good node that held a bad election.

From the possible violation of these three conditions, we lose no more than a $c'/\log n$ fraction of arrays that are good. Adding in good arrays lost from good elections, we lose no more than a total of $(c'+1)/\log n$ fraction of arrays that are good at any level, for a constant c'.

Initially a $2/3+\epsilon$ fraction of the arrays are good. As we have shown, at any given level, the fraction of arrays that are good decreases by at most $c'/\log n$. Thus, the fraction of arrays that are good at level ℓ^* is at least $2/3+\epsilon-c\ell^*/\log n = 2/3+\epsilon-c'/\log\log n$.

Hence there are wq arrays at the top of the election tree of which at least a $2/3$ fraction were generated by good processors, uniformly and independently at random. For each array, there is a block which is secret shared among the processors of the root node, so that for all but a $1/\log n$ fraction of the ℓ^*-shares, at least a $2/3$ fraction of ℓ-shares are held by good processors. Sequentially, the protocol exposes a word from each of the arrays. By Lemma 3, each word generated by a good processor is learned by $1-1/\log n$ fraction of good processors in the root (which contains all the processors). □

We now prove Theorem 2. Lemma 11 shows that *a.e.random-source* is an a.e.(s,t) random source with $s=\Theta(\log^{4+\delta} n)$ and $t \geq 2/3s$ with probability $1-1/n^{C_1}$ for any constant C_1. Its output can be used as input to *a.e.BA-with-random-source*. The correctness of *a.e.BA* follows from the correctness of *a.e.BA-with-random-source*. By Theorem 4 the probability of failure of *a.e.BA-with-random-source* is $1/2^{t-1}$ given *a.e.random-source* is correct. The probability that either algorithm fails is given by the union bound of $1/2^{t-1}+1/n^{C_1}$. Choosing C_1 appropriately this is less than$1/n^c$, for any c. The running time follows from Lemma 9.

4. A.E. BYZANTINE AGREEMENT TO BYZANTINE AGREEMENT

We call a processor *knowledgeable* if it is good and agrees on a message m. Otherwise, if it is good, it is *confused*. In this section, we assume that $(1/2+\epsilon)n$ processors are knowledgeable. We give an algorithm which uses a sequential (s,t) *a.e.random-source* which generates a sequence of s numbers in $[1,...,\sqrt{n}]$ described in Section 3.7 to produce everywhere agreement on m with private channels, proving Theorem 3.

In the following algorithm, the constant a depends on c and ϵ and will be specified in the proof of correctness.

4.1 Proof of correctness

We first prove the following lemma. In the proofs below, we refer to a loop as *good* if the number k generated in the loop was generated uniformly at random and is almost everywhere agreed on.

LEMMA 12. *Assume at the start of the protocol $(1/2+\epsilon)n$ good processors agree on a message m. Let c be any positive constant. Then after a single execution of a loop:*

(1) With probability at least $1-4/(\epsilon\log n)-1/n^c$, if the loop is good the protocol results in agreement on m; and

```
Algorithm 3 a.e.BA-to-BA
Repeat $s$ times:
(1) Each processor $p$ does the following in parallel:
    Randomly pick a set of $a\sqrt{n}\log n$ processors without replacement; for each of
    these processors $j$, randomly pick $i \in [1, \ldots, \sqrt{n}]$, with replacement, and send
    a request with label $i$ to processor $j$.
(2) Each processor $p$ decides on the next number $k$ in $[1, ..., \sqrt{n}]$ generated by
    a.e.random-source.
(3) For each processor $p$, if $p$ receives a request with label $i$ from $q$ and $i = k$ then
    if $p$ has not received more than $\sqrt{n}\log n$ such requests (it is not overloaded), $p$
    returns a message to $q$.
(4) Let $x_i$ be the number of messages returned to $p$ by processors sent the request
    label $i$. Let $i_{max}$ be an $i$ such that $x_i \geq x_j$ for all j. If the same message $m$ is
    returned by $(1/2 + 5\epsilon/16)a\log n$ processors which were sent the request label
    $i_{max}$ then $p$ decides $m$. Otherwise, $p$ is undecided.
```

(*2*) *With probability* $1 - 1/n^c$, *for any loop, every processor either agrees on* m *or is undecided.*

To prove Lemma 12 we first prove several other lemmas.

Lemma 13. *Let* $\delta \leq 1/2$ *be a fixed positive constant and let* S *be a set of at least* $(1/2 + \delta)n$ *good processors. Then, w.h.p., for any one loop, for every processor* p *and every request label* i, *at least* $(1/2 + \delta/2)a\log n$ *processors which are sent* i *by* p *are in* S, *and fewer than* $(1/2 - \delta/2)a\log n$ *processors which are sent* i *by* p *are not in* S. *This is true even if* S *is chosen by the adversary at any point during the execution of the loop.*

Proof. Since the channels are private, the adversary does not know p's requests other than those sent to bad processors. Hence the choice of S is independent of the queries received by bad processors.

Let X be the number of processors in S sent a request by processor p. X is a hypergeometric distribution and $E(X) = (1/2 + \delta)a\sqrt{n}\log n$. By concentration bounds on the hypergeometric (see e.g. [Janson 1999]), we know that: $Pr(X \leq E(X) - (\delta/8)a\sqrt{n}\log n) \leq 2e^{-(\delta/8a\sqrt{n}\log n)^2/2E(X)}$. This is less than n^{-c} for $a \geq 128c(1/2 + \delta)(\ln 2)/(\sqrt{n}\delta^2))$. Thus, for appropriate a, w.h.p., $X \geq (1/2 + (7/8)\delta)a\sqrt{n}\log n$.

Now for a given label i, let X_i be the number of processors in S that are sent a request with label i. Assuming that $X \geq (1/2 + (7/8)\delta)a\sqrt{n}\log n$, we know that $E(X_i) \geq \mu_L = (1/2 + (7/8)\delta)a\log n$. Moreover, X_i is the sum of i.i.d. random variables so by Chernoff bounds, $Pr(X_i \leq \mu_L - (\delta/8)a\log n) \leq e^{-((\delta/8)a\log n)^2/2\mu_L}$. This is less than n^{-c} for $a \geq 128c(1/2 + (7/8)\delta)(\ln 2)/\delta^2))$. Thus, for appropriate a, w.h.p., $X_i \geq (1/2 + (3/4)\delta)a\log n$.

Finally, let Y_i be the total number of requests with label i. We know that $E(Y_i) = a\log n$, and by Chernoff bounds, $Pr(Y_i \geq E(Y_i) + (\delta/4)a\log n) \leq e^{-((\delta/4)a\log n)^2/2E(X_i)}$. This is less than n^{-c} for $a = 32c(\ln 2)/\delta^2$. Thus, for appropriate a, w.h.p., $Y_i \leq (1 + \delta/4)a\log n$. Hence, for a fixed processor p and request label i, at least $(1/2 + \delta/2)a\log n$ processors which are sent i by p are in S, and fewer than

$(1/2 - \delta/2)a\log n$ processors which are sent i by p are not in S.

Taking a union bound over all labels i, we have that for fixed processor and for all labels, the lemma fails to hold with probability no more than $2n^{-c+1/2}$. Then, taking a union bound over all processors, we have that for all processors and for all labels, the lemma fails to hold with probability no more than $2n^{-c+3/2}$. Clearly, this probability of failure can be made smaller than any arbitrary polynomial for appropriate choice of c. $\Box$

The following corollary is immediate from Lemma 13 by letting S be the set of knowledgable processors and $\delta = \epsilon$.

Corollary 2. *Suppose there are at least $(1/2+\epsilon)n$ knowledgeable processors. W.h.p., for any one loop, for every processor p and every request label i, at least $A = (1/2+\epsilon/2)a\log n$ processors which are sent i by p are knowledgeable, and fewer than $B = (1/2-\epsilon/2)a\log n$ processors which are sent i by p are bad or confused.*

Corollary 2 immediately implies statement (2) of Lemma 12.

We need a couple of lemmas to establish the correctness of Lemma 12 (1). A knowledgeable processor p which is sent $i = k$ will respond unless overloaded. Each processor can receive no more than $n-1$ requests, since if two or more requests come from the same sender, that sender is evidently bad. Thus, there can be no more than $\sqrt{n}/\log n$ values of i for which there are more than $\sqrt{n}\log n$ requests labelled i. Then we claim:

Lemma 14. *The probability that more than $\epsilon n/4$ knowledgeable processors are overloaded in any one good loop of the algorithm is less than $4/(\epsilon\log n)$.*

Proof. We call a value i for a processor overloaded if $\sqrt{n}\log n$ request labels equal i. A processor is only overloaded if $k = i$ and the value i is overloaded. Since the loop is good, k is randomly chosen and each processor has at most a $1/\log n$ chance of being overloaded. Let X be a random variable giving the number of overloaded knowledgeable processors and y be the number of knowledgeable processors. Then $E[X] = y/logn$. Using Markov's Inequality, $Pr[X \geq y(\epsilon/4)] < (y/\log n)/(y\epsilon/4) = 4/(\epsilon\log n)$. $\Box$

We can now prove Lemma 12 (1). With probability at least $1 - 4/(\epsilon\log n)$, there are $(1/2+3\epsilon/4)n$ knowledgeable processors that are not overloaded. By the definition of an *a.e.random-source*, all but $n/\log n$, or all but $\epsilon n/8$ for sufficiently large n, processors will receive the same random value k in a good loop of the protocol. Thus, there is a set of at least $(1/2+5\epsilon/8)n$ processors that are knowledgeable, not overloaded and that received the same value k. Thus, setting S to be this set, and δ to be $5\epsilon/8$ in Lemma 13, we have w.h.p., for every processor and request label i that at least $A = (1/2+5\epsilon/16)n$ processors sent i by p are knowledgeable, not overloaded and received the same value k. Therefore, for any c, with probability at least $1 - 4/(\epsilon\log n) - 1/n^c$, a good loop of this protocol results in agreement on m.

We can now prove Theorem 3. The correctness of Algorithm *a.e.BA-to-BA* proceeds from Lemma 12 as follows: Set c to $c+1$ in Lemma 12. After s loops, including $t = ((c+1)/3)\epsilon\ln^2 n$ good loops, Algorithm 4 ends without agreement by some processor on m only if no good loop brings agreement on m, or some loop results in a bad decision. As each repetition of a loop is independent, the probability of the first type of failure is the product of the individual failures, or no more than $(1 - 4/(\epsilon \log n) - 1/n^c)^t = 1/n^{c+1}$, by Lemma 12 (1). The probability of the second type of failure is no greater than the sum of the individual failures or s/n^{c+1}, by Lemma 12 (2). Hence the probability of failure is less than the sum of the probabilities of these failures or $(s+1)/n^{c+1}$ which is less than $1/n^c$ for $s < n - 1$.

The running time of *a.e.BA-to-BA* is $f(s)$ per loop for a total of $O(sf(s))$. The number of bits is $\tilde{O}(g(s) + n^{1/2})$ per processor per loop or $\tilde{O}(s(g(s) + n^{1/2}))$. This concludes the proof of Theorem 3.

5. BYZANTINE AGREEMENT (*BA*)

Here we give an algorithm to compute (everywhere) Byzantine agreement:

Algorithm 4 *BA*

```
(1) Run a.e.BA to come to almost everywhere consensus on a bit $b$.
(2) Run a.e.BA-to-BA to ensure that all processors output $b$.
```

We next use this algorithm to prove Theorem 1.

5.1 Proof of Theorem 1

Proof. Let $\delta = 8$ in Theorem 2. By Theorem 2, there is a polylogarithmic time algorithm (*a.e.BA*) which uses $\tilde{O}(\sqrt{n})$ bits per processor to compute a.e. Byzantine agreement and an a.e. $(s, 2s/3)$ random source with $s = \Theta(\log^8 n)$ and $t = 2s/3$. Then $(1 - 1/\log n)(2/3 + \epsilon)n > (1/2 + \epsilon')n$ fraction of processors are good and knowledgeable, for $\epsilon' = 1/6$ and sufficiently large n. To generate a sequence s of numbers $\log n/2$ in length requires $g(s) = \tilde{O}(n^{4/\delta} s \log n/2) = \tilde{O}(n^{1/2})$ bits of communication and $f(s) = O(s \log n/2(\log n/\log\log n) = \tilde{O}(1)$ time. By Theorem 3, since t is $\omega(\log^2 n)$, then for any constants c and ϵ, and sufficiently large n, *a.e.BA-to-BA* ensures with probability $1 - 1/n^c$ that all processors output m in time $O(s + f(s)) = O(\log^8 n) + \tilde{O}(1) = \tilde{O}(1)$ time and $\tilde{O}(sn^{1/2} + g(s)) = \tilde{O}(n^{1/2})$ bits of communication per processor. This concludes the proof of Theorem 1. □

6. CONCLUSION

We have described an algorithm that solves the Byzantine agreement problem with each processor sending only $\tilde{O}(\sqrt{n})$ bits. Our algorithm succeeds against an adaptive, rushing adversary in the synchronous communication model. It assumes private communication channels but makes no other cryptographic assumptions. Our algorithm succeeds with high probability and has latency that is polylogarithmic in n. Several important problems remain including the following: Can we use $o(\sqrt{n})$ bits per processor, or alternatively prove that $\Omega(\sqrt{n})$ bits are necessary for agreement against an adaptive adversary? Can we adapt our results to the asynchronous communication model? Can we use the ideas in this paper to perform scalable, secure multi-party computation for other functions? Finally, can the techniques in this paper be used to create a practical Byzantine agreement algorithm for real-world, large networks?

REFERENCES

ABRAHAM, I., DOLEV, D., GONEN, R., AND HALPER, J. 2006. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Principles of Distributed Computing(PODC)*.

ABRAHAM, I., DOLEV, D., AND HALPER, J. 2008. Lower bounds on implementing robust and resilient mediators. In *IACR Theory of Cryptography Conference(TCC)*.

AGBARIA, A. AND FRIEDMAN, R. 2003. Overcoming Byzantine Failures Using Checkpointing. *University of Illinois at Urbana-Champaign Coordinated Science Laboratory technical report no. UILU-ENG-03-2228 (CRHC-03-14)*.

AMIR, Y., DANILOV, C., DOLEV, D., KIRSCH, J., LANE, J., NITA-ROTARU, C., OLSEN, J., AND ZAGE, D. 2006. Scaling Byzantine fault-tolerant replication to wide area networks. In *Proc. Int. Conf. on Dependable Systems and Networks*. Citeseer, 105–114.

ANDERSON, D. AND KUBIATOWICZ, J. 2002. The worldwide computer. *Scientific American 286,* 3, 28–35.

CASTRO, M. AND LISKOV, B. 2002. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS) 20,* 4, 398–461.

CHENG, C.-F., WANG, S.-C., AND LIANG, T. 2009. The anatomy study of server-initial agreement for general hierarchy wired/wireless networks. *Computer Standards & Interfaces 31,* 1, 219 – 226.

CHOR, B., GOLDREICH, O., HÅSTAD, J., FRIEDMAN, J., RUDICH, S., AND SMOLENSKY, R. 1985. The bit extraction problem of t-resilient functions (preliminary version). In *FOCS*. IEEE, 396–407.

CLEMENT, A., MARCHETTI, M., WONG, E., ALVISI, L., AND DAHLIN, M. 2008. Byzantine fault tolerance: the time is now. In *LADIS '08: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*. ACM, New York, NY, USA, 1–4.

CLEMENT, A., WONG, E., ALVISI, L., DAHLIN, M., AND MARCHETTI, M. 2009. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*.

COWLING, J., MYERS, D., LISKOV, B., RODRIGUES, R., AND SHRIRA, L. 2005. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In *In Proceedings of Operating Systems Design and Implementation (OSDI)*. San Diego, CA, USA.

DOLEV, D. AND REISCHUK, R. 1985. Bounds on information exchange for byzantine agreement. *J. ACM 32,* 1, 191–204.

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

GARAY, J. A. AND OSTROVSKY, R. 2008. Almost-everywhere secure computation. In *EUROCRYPT*. 307–323.

HOLTBY, D., KAPRON, B. M., AND KING, V. 2008. Lower bound for scalable byzantine agreement. *Distributed Computing 21,* 4, 239–248.

JANSON, S. 1999. On concentration of probability. *Combinatorics, Probability and Computing 11*, 2002.

KAPRON, B., KEMPE, D., KING, V., SAIA, J., AND SANWALANI, V. 2009. Scalable algorithms for byzantine agreement and leader election with full information. *ACM Transactions on Algorithms(TALG)*.

KING, V., OLUWASANMI, O., AND SAIA, J. 2010. An empirical study of a scalable byzantine agreement aglrotihm. In *19th International Heterogeneity in Computing Workshop*.

KING, V. AND SAIA, J. 2009. From almost-everywhere to everywhere: Byzantine agreement in $\tilde{O}(n^{3/2})$ bits. In *International Symposium on Distributed Computing (DISC)*.

KING, V., SAIA, J., SANWALANI, V., AND VEE, E. 2006a. Scalable leader election. In *Proceedings of the Symposium on Discrete Algorithms(SODA)*.

KING, V., SAIA, J., SANWALANI, V., AND VEE, E. 2006b. Towards secure and scalable computation in peer-to-peer networks. In *Foundations of Computer Science(FOCS)*.

KOTLA, R., ALVISI, L., DAHLIN, M., CLEMENT, A., AND WONG, E. 2007. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. ACM, 58.

Malkhi, D. and Reiter, M. 1997. Unreliable intrusion detection in distributed computations. In *In Computer Security Foundations Workshop*. 116–124.

McEliece, R. J. and Sarwate, D. V. 1981. On sharing secrets and reed-solomon codes. *Commun. ACM 24,* 9, 583–584.

Rabin, M. 1983. Randomized byzantine generals. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*. 403–409.

Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., and Kubiatowicz, J. 2003. Pond: the OceanStore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*. 1–14.

Shi, E., Perrig, A., et al. 2004. Designing secure sensor networks. *IEEE Wireless Communications 11,* 6, 38–43.

Wright, A. 2009. Contemporary approaches to fault tolerance. *Communications of the ACM 52,* 7, 13–15.

Yoshino, H., Hayashibara, N., Enokido, T., and Takizawa, M. 2005. Byzantine agreement protocol using hierarchical groups. In *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems*. IEEE Computer Society, Washington, DC, USA, 64–70.

Received February 1986; November 1993; accepted January 1996