Byzantine Agreement in Expected Polynomial Time

Valerie King, University of Victoria
Jared Saia, University of New Mexico

We address the problem of Byzantine agreement, to bring processors to agreement on a bit in the presence of a *strong* adversary. This adversary has full information of the state of all processors, the ability to control message scheduling in an asynchronous model, and the ability to control the behavior of a constant fraction of processors which it may choose to corrupt adaptively.

In 1983, Ben-Or proposed an algorithm for solving this problem with expected exponential communication time. In this paper, we improve that result to require expected polynomial communication time and computation time. Like Ben-Or's algorithm, our algorithm uses coinflips from individual processors to repeatedly try to generate a fair global coin. We introduce a method which uses spectral analysis to identify processors that have thwarted this goal, by flipping biased coins.

Categories and Subject Descriptors: [**Theory of Computation**]: Distributed Algorithms

General Terms: Algorithms, Design, Reliability, Theory

Additional Key Words and Phrases: Byzantine agreement, Spectral Methods, Global Coin, Security, Randomization

ACM Reference Format:
Valerie King and Jared Saia, 2014. Faster Agreement via a Spectral Method for Detecting Malicious Behavior *J. ACM* 0, 0, Article 0 (2014), 22 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

How can we build a reliable system when many parts are unreliable? Byzantine agreement is fundamental to addressing this question. The Byzantine agreement problem is to devise an algorithm so that n processors, each with a private input can agree on a single common output that is equal to some processor's input. For example, if all processors start with 1, they must all decide on 1. The processors should successfully terminate despite the presence of $t = \Theta(n)$ bad processors. An adversary controls the behavior of the bad processors which can deviate from the algorithm in arbitrary ways. Byzantine agreement is one of the most fundamental problems in distributed computing; it has been studied for over 30 years and is referenced in tens of thousands of papers.

In this paper, we consider Byzantine agreement in the challenging *classic* asynchronous message-passing model. There is an authenticated channel between every pair of processors. The adversary is *adaptive*: it can determine which processors to cor-

Valerie King was partially supported by an NSERC grant, PIMS, Princeton IAS and Simons Institute Berkeley. Jared Saia was partially supported by NSF CAREER Award 0644058 and NSF CCR-0313160.
Author's addresses: V. King, Department of Computer Science University of Victoria, Victoria, B.C. CANADA V8W 3P6 and J. Saia, Department of Computer Science, University of New Mexico, Farris Engineering Building, Albuquerque, NM 87131-1386.

DOI: http://dx.doi.org/10.1145/0000000.0000000

rupt and what strategy these processors should use as the algorithm proceeds. Communication is *asynchronous*: the scheduling of the delivery of messages is set by the adversary, so that the delays are unpredictable to the algorithm. Finally, the adversary has *full information*: it knows the states of all processors at any time, and is assumed to be computationally unbounded. Such an adversary is also known as "strong" [Attiya and Welch 2004].

We assume that each processor has its own fair coin and may at any time flip the coin and decide what to do next based on the outcome of the flip. The major constraint on the adversary is that it cannot predict future coinflips.

Communication time in this model is defined to be the maximum length of any chain of messages (see [Chor and Dwork 1989; Attiya and Welch 2004]). In particular, all computation by individual processors is assumed to be instantaneous, and sending a message over the network is counted as taking 1 unit of time. In addition, we consider *computation time* by individual processors, which is measured in the usual way.

The only results known to the authors for this classic model are the works of Ben-Or (1983) [Ben-Or 1983] and Bracha (1984) [Bracha 1984; 1987a]. Ben-Or gave a Byzantine agreement (BA) algorithm tolerating $t < n/5$. Bracha improved this tolerance to $t < n/3$. Unfortunately, both of these algorithms run in exponential expected communication time if $t = \Theta(n)$. In 2006, Ben-Or, Pavlov and Vaikuntanathan [Ben-Or et al. 2006] wrote:

"In the case of an asynchronous network, achieving even a polynomial-rounds BA protocol is open. We note that the best known asynchronous BA protocols [Ben-Or 1983; Bracha 1984] have exponential expected round-complexity"

To the authors' knowledge, we present the first algorithm for this problem to achieve better than exponential expected run time.

1.1. Main Results

The main results of this paper are as follows.

THEOREM 1.1. *Byzantine Agreement can be solved in the classic model in expected $O(n^3)$ communication time, expected polynomial computation time per processor, and expected polynomial bits of communication. The number of faulty processors that can be tolerated is linear, specifically $t < 4.25 * 10^{-7} n$.*

We can improve communication time by using exponential computation time as follows.

THEOREM 1.2. *Byzantine Agreement can be solved in the classic model in expected $O(n^{2.5}\sqrt{\log n})$ communication time, expected exponential computation time per processor, and expected polynomial bits of communication. The number of faulty processors that can be tolerated is linear, specifically $t < n/400$.*

Results here first appeared in conference proceedings [King and Saia 2013] and [King and Saia 2014].

1.2. Other Contributions

Our algorithm involves repeated attempts to compute a global coin. We show that processors which successfully thwart this goal must engage in behavior that is detectable using spectral analysis. This suggests a new paradigm for secure distributed computing: the design of algorithms such that an effective attack exposes the identity of processors controlled by the adversary.

We also define the asynchronous *blackboard model* to simplify the presentation of our algorithms. We show it can be implemented with only constant factor overhead in

the classic model. The blackboard model depicts all communication among processors as a sequence of matrices. Once a matrix is generated, all processors $sync$, i.e., they read all the same values except perhaps one for each of t processors, before generating the next matrix. We believe that the asynchronous blackboard model may be of independent interest for developing algorithms in the classic model.

Paper Organization: The rest of this paper is organized as follows. Section 2 describes related work. In Section 3, we present our main algorithm. It is a modified version of Bracha's algorithm, MODIFIED-BRACHA, which calls upon a coinflip algorithm we call GLOBAL-COIN. The blackboard model and the GLOBAL-COIN procedure are presented in Section 4.

In Section 5, we analyze properties of the coinflips generated and broadcast over multiple calls to GLOBAL-COIN. In particular, we perform an analysis after $\Theta(n)$ calls (an "epoch") to GLOBAL-COIN.

We then describe two different methods for eliminating bad processors. In Section 6, we describe PROCESS-EPOCH 1 which uses techniques from spectral analysis, and has expected polynomial computation time and expected $O(n^3)$ communication time. In Section 7, we describe PROCESS-EPOCH 2, which uses a brute-force approach, and has expected exponential computation time and expected $O(n^{2.5} \log n)$ communication time.

Section 8 discusses future directions and open problems.

2. RELATED WORK

2.1. Byzantine agreement

Papers published on Byzantine agreement number into the thousands. We refer the reader to [Attiya and Welch 2004; Lynch 1996] for a general overview of the problem. For conciseness, we focus here only on the classic asynchronous model, where the adversary is adaptive and has full information.

The Byzantine agreement problem was introduced over 30 years ago by Lamport, Shostak and Pease [Lamport et al. 1982]. In the model where faulty behavior is limited to adversary-controlled stops known as *crash failures*, but bad processors otherwise follow the algorithm, the problem of Byzantine agreement is known as *consensus*. In 1983, Fischer, Lynch and Paterson (FLP) showed that a deterministic algorithm cannot solve the consensus problem in an asynchronous model even with one faulty processor [Fischer et al. 1985].

In 1983, Ben-Or introduced randomization, where each processor can flip a random private coin, as a way to avoid the FLP impossibility result. His algorithm solved Byzantine agreement in communication time exponential in the number of processors, in the classic asynchronous model. His algorithm consists of multiple rounds in which each good processor flips a coin. The communication time is proportional to the expected number of rounds before the number of heads exceeds the number of tails by more than t. Thus, in expectation, this algorithm has constant running time if $t = O(\sqrt{n})$, but has exponential running time for t any constant fraction of n, up to $t < n/5$.

The resilience (number of faulty processors tolerated) was improved to $t < n/3$ by Bracha [Bracha 1984; 1987a]. The communication time remained exponential. This resilience is the best possible even for randomized algorithms[Karlin and Yao 1984].

2.1.1. Variants on Classic Model. We note that the full-information assumption makes the problem challenging. Randomized algorithms for Byzantine agreement with resilience $t = \Theta(n)$ and constant expected time have been known for 25 years, under the assumption of private channels (so that the adversary cannot see messages passed between processors) [Feldman and Micali 1985]. Under these assumptions, more recent

work shows that optimal resilience can be achieved [Canetti and Rabin 1993]. With cryptographic assumptions, it is possible to achieve Byzantine agreement in $O(1)$ communication time and polynomial computation time, even in the asynchronous model when the adversary is adaptive (see e.g. [Cachin et al. 2005]).

Byzantine agreement was also more recently shown to require only polylogarithmic time in the full information model if the adversary is *static*, meaning that the adversary must choose the faulty processors at the start, without knowing the random bits of the algorithm [Kapron et al. 2009]. The technique is to elect a very small subset of processors which contain a less than 1/3 fraction of faulty processors; this subset then runs the exponential time algorithm on their inputs. Such a technique does not seem applicable when the adversary is adaptive and can decide to corrupt the elected set after it sees the result of the election.

2.1.2. Lower Bounds. Randomized consensus (and Byzantine agreement) algorithms are discussed extensively in Aspnes's 2003 survey on the topic [Aspnes 2003]. That paper includes a discussion of the solution to consensus in the shared memory model when up to $n-1$ crash failures occur. In the shared memory model, cost is measured by the total step complexity. Algorithms with expected polynomial steps for consensus with shared memory were introduced in 1990 [Aspnes and Herlihy 1990].

In 1998, Aspnes showed a $\Omega(n^2/\log^2 n)$ bound on the number of coinflips required for consensus [Aspnes 1998]. In 2008, Attiya and Censor-Hillel showed tight upper and lower bounds of $\Theta(n^2)$ on the total number of steps required for consensus in the shared memory model [Attiya and Censor 2008]. Any shared memory algorithm for consensus can be simulated by a message passing algorithm with constant time overhead, provided that the number of faults is less than $n/2$. The lower bound of $\Omega(n^2)$ steps implies a $\Omega(n)$ time bound for consensus[1] in the message passing model, where at least $n/2$ processors may be executing in parallel.

In 2011, Lewko [Lewko 2011] considered a certain class of "fully symmetric round protocols" for solving Byzantine agreement in the asynchronous model with an an adaptive adversary with full information. In a fully symmetric protocol, "a processor computes its message to broadcast in the next round as a randomized function of the set of messages it has validated, without regard to their senders." Lewko showed that any such protocol could be forced by an adversary to run in exponential expected time. Since our algorithm considers the identity of processors when processing coinflips, our algorithm is not a fully symmetric round protocol.

2.2. Blackboard Model

Our blackboard model can be viewed as an extension of the Spread algorithm of [Ben-Or and El-Yaniv 2003], and the algorithms to simulate identical Byzantine failure that are due to Neiger and Toueg [Neiger and Toueg 1990] and Bracha [Bracha 1987b] (see also Chapter 12 of [Attiya and Welch 2004]).

2.3. Spectral Methods

Spectral methods have been used frequently to identify trustworthy and untrustworthy processors in decentralized systems. Perhaps one of the most prominent applications is identifying trustworthy web pages. The PageRank algorithm [Page et al. 1999] (which was inspired by the spectral-based Clever algorithm of Kleinberg [Kleinberg 1999; Chakrabarti et al. 1999]) is well-known as the basis by which Google ranks web documents. PageRank ranks web pages based on the top eigenvector of a stochastic matrix that describes a random walk over the web graph. This eigenvector corresponds

[1] as well as Byzantine agreement

to the stationary distribution of the random walk, and pages that have high probabilities in this stationary distribution are considered to be "authoritative" web pages. It is known that PageRank is relatively robust to adversarial attempts to thwart it by adding a small number of spurious links to the web graph [Zhang et al. 2004; Azar et al. 2001].

The idea of PageRank is the basis of the eigentrust algorithm [Kamvar et al. 2003] (see also [Xiong and Liu 2004; Guha et al. 2004; Zhang et al. 2004]). Eigentrust calculates the stationary distribution (the top eigenvector) of a random walk in a trust graph, where an edge from processor i to processor j has weight $w_{i,j}$ that indicates how much processor i trusts processor j. Processors with high probabilities in this stationary distribution are considered trustworthy. Eigentrust also provides some protection against collusion by bad processors.

We note that, in a sense, our approach is the opposite of eigentrust. In our algorithm, processors with high absolute values in the top right singular vector are *not* trustworthy. Intuitively, this is because in our algorithm, good processors have random coinflips, and so over time, the columns controlled by these processors will have little "structure", which translates to a small absolute value in the singular vector.

Our spectral approach shares some similarities with the hidden clique detection problem. In this problem, proposed independently by Jerrum [Jerrum 1992] and Kucera [Kučera 1995], a random $G(n, 1/2)$ graph is chosen and then a clique of size k is randomly placed in the graph. Alon, Krivelivich and Sudakov [Alon et al. 1998] described a spectral algorithm that can find a clique, with high probability ("w.h.p.") i.e., $1 - 1/n^c$ for any fixed constant c, when $k = \Omega(\sqrt{n})$ [Alon et al. 1998]. Roughly, this algorithm 1) finds the second eigenvector of the adjacency matrix of this graph; 2) sets W to be the top k vertices when the vertices are sorted in decreasing order by their absolute value in this eigenvector; and 3) returns as the clique the set of all vertices of the graph that have at least $3k/4$ neighbors in W.

3. THE BYZANTINE-AGREEMENT ALGORITHM

In this section, we describe our main algorithm BYZANTINE-AGREEMENT (Algorithm 1). We first give an overview of the algorithm.

3.1. Algorithm Overview

Our algorithm BYZANTINE-AGREEMENT proceeds in epochs. Each epoch consists of $m = \Theta(n)$ iterations of MODIFIED-BRACHA, which is based on Bracha's 1984 exponential expected time Byzantine agreement algorithm. Each iteration of MODIFIED-BRACHA attempts to generate a global coin using the algorithm GLOBAL-COIN. In each iteration, there is a "correct direction" that is unknown to the algorithm. If the global coin is in the correct direction for enough good processors, then Algorithm 1 will succeed in the next iteration.

Initially, each processor p has a whitelist, V_p, consisting of all processors. When processor p calls GLOBAL-COIN, it uses coinflip messages only from processors in V_p. If, for a constant fraction of processors p, V_p consists of all good processors, then with constant probability, the global coin will be in the correct direction, and the algorithm will terminate in the next iteration.

The challenge for a processor p is to analyze coinflip messages received during an epoch to decide which processors to remove from V_p. This is the purpose of PROCESS-EPOCH. There are two variants of PROCESS-EPOCH, a spectral variant (Section 6) and an exponential computational time variant (Section 7).

In both variants of PROCESS-EPOCH, each processor p analyzes data from the epoch to assign "badness" in the form of $\mathbf{score}_p(v)$ to processors v. When the score value for a processor gets sufficiently high, that processor is removed from V_p. Only

coinflips from processors in V_p are used by p in future epochs. After $c_1 n$ epochs, where c_1 is a constant, if no agreement is reached, the sets V_p are reinitialized to V, and the algorithm is restarted from scratch.

A key technical detail remains: how to deal with asynchronicity. Our main tool for this is the asynchronous blackboard model (Section 4). In this model, there is an idealized m by n matrix in which each of the n processors writes up to n values. We describe an algorithm, x-synch (Section 4.1, that ensures each processor will read all the same values from this matrix, except perhaps one value for each of t processors. We use the blackboard model to ensure that in each iteration, the adversary is not able to use asynchronous message scheduling to throw out too many coinflip messages from good processors.

In the rest of this section, we discuss details of MODIFIED-BRACHA (Section 3.3) and Bracha's Reliable Broadcast algorithm (Section 3.2), which is used as a primitive in MODIFIED-BRACHA, and in many other algorithms throughout this paper.

ALGORITHM 1: BYZANTINE-AGREEMENT

```
1: while there is no decision do
2:    For each v ∈ V, score_p(v) ← 0
3:    V_p ← the set of all processors
4:    for e = 1 to c_1 n {"p runs epoch e"} do
5:       for i = 1 to m do
6:          Run iteration i of MODIFIED-BRACHA
7:       end for
8:       Run PROCESS-EPOCH
9:    end for
10: end while
```

3.2. Reliable Broadcast

ALGORITHM 2: Bracha's Reliable Broadcast

1: Send $(initial, m)$ to all processors.
2: Upon receiving $(initial, m)$, send $(echo, m)$ to all processors.
3: Upon receiving either $n - t$ messages $(echo, m)$ or $t + 1$ $(ready, m)$ messages, send $(ready, m)$ to all processors.
4: Upon receiving $n - t$ $(ready, m)$ messages, r-receive m.

Our algorithm makes use of *reliable broadcast* which we will use throughout this paper. In this constant round primitive, a single player calls *reliable broadcast* [Bracha 1987a] for a particularly message m, and subsequently, all players may *r-receive* exactly one message.

Our version of reliable broadcast Algorithm 2 has the following properties.

Reliable Broadcast Properties

(1) If a good player broadcasts a message m, then all good players eventually r-receive m.
(2) If a bad player p broadcasts a message then either all good players r-receive the message, or no good players r-receive the message.
(3) If a message is r-received, then at least $n - 2t$ good processors participated in the broadcast.

While Property 3 is not typically stated, it is easy to see[2], and is used in the proof of Lemma 6 in [Ben-Or and El-Yaniv 2003].

Note: When a reliable broadcast is initiated by a processor, all processors continue to participate in this reliable broadcast while executing other steps of the algorithm.

The next lemma describes the latency of this algorithm.

LEMMA 3.1. *If a good processor reliably broadcasts a message, the latency of that reliable broadcast is* $O(1)$*. If a bad processor reliably broadcasts a message, then if* 1 *processor r-receives a message, then all good processors will r-receive that message in* 2 *units of time.*

PROOF. To see this, note that if a good processor r-received a message m at time x then it received $n-t$ $(ready, m)$ messages at time x. Thus, $n-2t$ good processors sent $(ready, m)$ messages by time x. Then, every good processor will receive $n-2t \geq t+1$ $(ready, m)$ messages by time $x+1$ and will send $(ready, m)$ messages. Hence at time $x+2$, all good processors will receive $n-t$ $(ready, m)$ messages and so will r-receive m. □

3.3. MODIFIED-BRACHA

We refer to each iteration of the while-loop as an *iteration* of MODIFIED-BRACHA. We assume every message is labelled with its step number and iteration number.

We say that processor p *validates* a message as follows. It validates all messages r-received in step 1. It validates a message m from processor q in step $k > 1$ if there exists a subset of messages validated by p from steps less than k which if received by q could cause q to send m in step k.

In Bracha's 1984 algorithm, a global coin is created, with exponentially small probability, when each processor flips one coin and their values all coincide. We replace these individual coin flips with a routine, GLOBAL-COIN involving up to n coin flips per processor.

Note that some processors may participate in GLOBAL-COIN even though they do not use its outcome, to ensure full participation by good processors. In MODIFIED-BRACHA, v_p is initialized to be the processor p's input bit for Byzantine agreement.

ALGORITHM 3: MODIFIED-BRACHA for process p

1: **while** there is no decision, repeat **do**
2: Reliably broadcast v_p and wait until validate $n-t$ messages;
3: $v_p \leftarrow$ majority of the r-received messages.
4: Reliably broadcast v_p and wait until validate $n-t$ messages;
5: If more than $n/2$ messages have same value v then $v_p \leftarrow v$
6: Reliably broadcast v_p and wait until validate $n-t$ messages;
7: Let $x =$ number of messages of the form v are r-received.
8: a) CASE $x > 2t$: decide v;
9: b) CASE $t < x \leq 2t$: run GLOBAL-COIN but set $v_p \leftarrow v$;
10: c) CASE $x \leq t$: $v_p \leftarrow$ GLOBAL-COIN.
11: **end while**

LEMMA 3.2 ([BRACHA 1987A]). *In an iteration of* MODIFIED-BRACHA *with* $t < n/3$*:*

[2]If less than $n-2t$ good processors participate, then the last step of Algorithm 2 will never be completed

(*1*) *Either there is agreement in the current or next iteration, or all good processors run* GLOBAL-COIN.
(*2*) *If greater than* $2n/3$ *good processors start with the same bit value* v*, then every good processor will decide on* v *in that iteration. In particular, if* GLOBAL-COIN *returns* v *(from Step 4(b)) to* $2n/3$ *good processors then every good processor comes to agreement in the next iteration.*

4. THE BLACKBOARD MODEL

The goal of GLOBAL-COIN is to generate, with constant probability, a fair coinflip that is agreed upon by a large fraction of good processors, or to provide data which, after $\Omega(n)$ iterations, will enable individual processors to identify bad processors. The algorithm requires each processor to repeatedly perform a coinflip where heads is +1 and tails is -1, and broadcast up to n of these coinflips. Upon receiving sufficiently many coinflips, each processor p computes the sum of coinflips received from each processor $q \in V_p$, and then decides on the sign of the total sum of coinflips received. In order to implement GLOBAL-COIN, we introduce a new model, the asynchronous blackboard model, which may be of independent interest.

4.1. The x-synch algorithm

The blackboard model where each player writes to a blackboard that all can see is a model commonly used in the study of communication complexity. Here, we describe an asynchronous blackboard model which can be simulated in our message-passing model with t processors controlled by the adversary.

The modified blackboard model runs in global time steps called x-syncs where x is a parameter specified by the algorithm ahead of time. It has the following properties. Each x-synch consists of up to x individual processor time steps. The communication which results from an x-sync is described by what we call an *asynchronous blackboard* denoted by $\mathcal{BB}$. $\mathcal{BB}$ is defined globally, but each processor j may have a different view of $\mathcal{BB}$, denoted $\mathcal{BB}_j$.

An x-synch ensures that 1) at least $n-t$ processors write x messages and these messages are read by all processors; 2) all remaining processors p write up to x messages and all of p's messages, except possibly the last, are read by all other processors; and 3) the adversary must decide whether or not to allow a write to the blackboard by a processor at time t before the processor initiates a write at time $t+1$.

A 1-synch is more powerful than a usual "round" in an asynchronous message passing model in the following way. A typical round ends for a processor when it receives $n-t$ messages from *any* subset of processors in that round. A 1-synch is stronger in that it ensures that there is a set of $n-t$ processors such that *all* processors read messages from this set before termination. [3]

We next give a detailed definition.

Properties of an x-synch For an x-synch with n processors, the asynchronous blackboard $\mathcal{BB}$ has n columns and m rows. We say that a value is *written at time* t to $\mathcal{BB}$ if after time t, the adversary cannot prevent that value from being in $\mathcal{BB}$.

If a cell has no value written to it, we say that that cell is null. A cell of the blackboard that is not null can be *ambiguous*, meaning that the value written to the cell can be seen by some processors, but the cell appears as null to others. If a cell is not ambiguous, then all processors read the same value at that cell when they terminate. Every processor eventually terminates the x-synch.

[3] A 1-synch is a slightly extended version of the protocol *Spread* as described by Ben-Or and El-Yaniv [Ben-Or and El-Yaniv 2003].

Recall that the adversary can take over up to t processors. Once the adversary takes over a processor, it controls it for the remainder of the algorithm. The adversary decides which processor to take over at any time during the algorithm and knows the state of all processors. An x-synch has the following key properties.

x-sync Properties:

(1) For all $1 < i \leq x$, each processor j generates a value for $\mathcal{BB}[i, j]$ only after writing $\mathcal{BB}[i-1, j]$
(2) There are $n - t$ columns, S, each containing x non-null entries on the blackboard. For every column $j \notin S$, there is a row $last(j)$ such that cells in $\mathcal{BB}[1, j], ..., \mathcal{BB}[last(j) - 1, j]$ have values written to them and are not ambiguous; the cell $\mathcal{BB}[last(j), j]$ has a value written to it, but is ambiguous; and cells $\mathcal{BB}[last(j+1), ..., \mathcal{BB}[m, j]$ are null.
(3) The adversary may see all messages generated by good processors during an x-synch before writing the messages of the processors it controls.

Algorithm 4 performs an x-synch in our message passing model. We let *message*(i, j) be the i-th message that processor j generates. For a processor j, the matrix $\mathcal{BB}_j$ is the output of the x-synch. We show later that for any entry (i, j) and any pair of processors x and y, if both $\mathcal{BB}_x(i, j)$ and $\mathcal{BB}_y(i, j)$ are not null, then $\mathcal{BB}_x(i, j) = \mathcal{BB}_y(i, j)$.

Hence we can define:

$$\mathcal{BB}(i,j) = \begin{cases} \mathcal{BB}_{j'}(i,j) & \text{if } \mathcal{BB}_{j'} \neq null \text{ for some } j' \\ \text{null} & \text{Otherwise} \end{cases}$$

"Participation" in reliable broadcast means to send messages in response to others' messages but not to initiate a reliable broadcast. We assume that all processors participate in reliable broadcasts initiated by others except where stated.

4.2. Analysis of the x-synch algorithm

LEMMA 4.1. *Every processor eventually terminates in Algorithm 4.*

PROOF. We first show that all processors leave the while-loop. We do this in two parts. First, we show that if one good processor leaves the while-loop, then eventually all good processors leave the while-loop. The one good processor must have r-received *message*$(1, j'), ...,$ *message*(x, j') from $n - t$ processors j'. By the property of reliable broadcast, every good processor will thus eventually r-receive these same messages.

We now show that one good processor will eventually leave the while-loop. Assume no good processor ever leaves the while-loop. For each processor j let i_j be the last i value set by processor j. There are two cases.

Case 1: At least one good processor j has $i_j < x$. Processor j has reliably broadcast *message*(i_j, j). Hence either $i_j = 1$ or it has r-received $n - t$ acks for *message*$(i_j - 1, j)$. Hence, these $n - t$ acks will eventually be r-received by all good processors. Thus, all $n - t$ good processors will eventually participate in the reliable broadcast of *message*(i_j, j). By assumption, no good processors leave the while-loop, thus all $n - t$ good processors will eventually r-receive *message*(i_j, j), and reliably broadcast an ack. Finally, $n - t$ acks will eventually be received by processor j for this message. Thus, processor j will increment i_j, which is a contradiction.

Case 2: For all good processors j, $i_j = x$. Then all the $n - t$ good processors j have reliably broadcast *message*$(1, j), ...,$ *message*(x, j). Then every good processor will r-receive these messages and will leave the while-loop and execute step 9.

ALGORITHM 4: x-synch for each processor j

1: $i \leftarrow 1$

{*Generate Messages:* }
2: **While** there are fewer than $n - t$ processors j' such that *message*$(1, j'), ...,$ *message*(x, j') have been r-received **do in parallel**
3: Generate and reliably broadcast *message*(i, j). Participate in reliable broadcast for *message*(i, j') for any processor j', only if $i = 1$ or have r-received $n - t$ acks for *message*$(i - 1, j')$.
4: Upon r-receiving *message*(i, j') for any processor j', reliably broadcast an ack for *message*(i, j')
5: **if** acks for *message*(i, j) are r-received from $n - t$ processors and $i < x$ **then**
6: increment i
7: **end if**
8: **end while**

{*Spread:* }
9: Reliably broadcast the matrix $\hat{\mathcal{BB}}_j$ where $\hat{\mathcal{BB}}_j(i, j') =$ *message*(i, j') if *message*(i, j') has been r-received and null otherwise. Participate in the reliable broadcast of $\hat{\mathcal{BB}}_{j"}$ for other processors j" only if the matrix contains $n - t$ full columns and each entry in the matrix has been r-received.
10: Wait until matrices $\hat{\mathcal{BB}}_{j"}$ are r-received from $t + 1$ processors j''
11: Update $\hat{\mathcal{BB}}_j$: replace any null entry $\hat{\mathcal{BB}}_j(i, j')$ by the non-null *message*(i, j') which has been r-received.

{*Final Update:* }
12: Reliably broadcast $\hat{\mathcal{BB}}_j$
13: Wait until r-receive matrices $\hat{\mathcal{BB}}_{j"}$ from $n - t$ processors j''
14: Update $\mathcal{BB}_j$: replace any null entry $\mathcal{BB}_j(i, j')$ by the non-null value in $\hat{\mathcal{BB}}_{j"}(i, j')$ if it appears in the $\hat{\mathcal{BB}}_{j"}$ for $t + 1$ processors j".

We have shown that all good processors eventually leave the while loop.

Now, let j be any good processor and $\hat{\mathcal{BB}}_j$ be the matrix reliably broadcast during step 9. $\hat{\mathcal{BB}}_j$ has $n - t$ columns where all entries have been r-received by j. Hence, each entry in these $n - t$ columns of $\hat{\mathcal{BB}}_j$ will eventually be r-received by all good processors.

Then all good processors will eventually participate in the reliable broadcast of $\hat{\mathcal{BB}}_j$, and so all good processors will eventually r-receive $\hat{\mathcal{BB}}_j$, and complete step 10, and then enter step 12. As all good processors will enter step 12, they will complete steps 13 and 14 and terminate. □

Correctness: We now show that Algorithm 4 satisfies the properties of an x-synch. We assume $t < n/4$.

LEMMA 4.2. *If* *message*(i, j) *is r-received by at least* $n - 2t$ *good processors before they execute step 12, then* $\mathcal{BB}[i, j]$ *is not ambiguous and not null.*

PROOF. Let *message*(i, j) be any message that is r-received by $n-2t$ good processors before they execute step 12. When any good processor, j' finishes step 13, it must have r-received matrices from $n-t$ processors, which includes at least $n-2t$ good processors, of which all but at most t have seen *message*(i, j) before broadcasting their matrices. Therefore, at least $n - 3t$ of the matrices received by processor j' in step 13 contain the value *message*(i, j) in the (i, j)-th entry of the matrix. Hence if $n - 3t > t$, processor j'

will set $\mathcal{BB}_{j'}[i,j]$ to *message*(i,j) in Step 14. Since j' was arbitrary, all good processors j' will do the same. □

LEMMA 4.3. *At least $n-t$ full columns of $\mathcal{BB}$ have no entries that are ambiguous or null.*

PROOF. In step 10, $t+1$ matrices are r-received by processor j; there must be at least one matrix which was broadcast by a good processor. Let $\hat{\mathcal{BB}}_{j'}$ be the first such matrix which is r-received by any good processor and t be the time it was r-received. Since j' has exited the while-loop, $\hat{\mathcal{BB}}_{j'}$ contains $n-t$ full columns of entries which j' has r-received. Since $\hat{\mathcal{BB}}_{j'}$ is r-received, by Property 3 of reliable broadcast, $n-2t$ good processors must have participated in the reliable broadcast of $\hat{\mathcal{BB}}_{j'}$ before time t. As specified in step 9, these processors must have r-received the entries of $\hat{\mathcal{BB}}_{j'}$ at some time $t' < t$. Since $\hat{\mathcal{BB}}_{j'}$ is the first such matrix, t' is prior to the time that any good processor finishes step 10.

Hence in step 11, for at least $n-2t$ good processors j", $\hat{\mathcal{BB}}_{j"}$ will match the non-null values of $\hat{\mathcal{BB}}_{j'}$.

Thus, by Lemma 4.2, $\mathcal{BB}[i,j]$ is not ambiguous and not null, for all non-null entries (i,j) of $\hat{\mathcal{BB}}_{j'}$. This includes the $n-t$ full columns of $\hat{\mathcal{BB}}_{j'}$. □

LEMMA 4.4. *In the matrix $\mathcal{BB}$, all entries in a column that follow an ambiguous entry are null, and all entries that precede it are non-null and non-ambiguous.*

PROOF. Fix a processor j' and let i' be the smallest value such that $\mathcal{BB}[i',j']$ is ambiguous or null. By the contrapositive of Lemma 4.2, fewer than $n-2t$ good processors r-received *message*(i',j') before they executed step 12. Thus, fewer than $n-2t$ processors r-received *message*(i',j') before they terminated the while-loop. Therefore, fewer than $n-2t$ good processors broadcast an ack for *message*(i',j') and no good processor participated in a reliable broadcast of any *message*(i'',j') for $i'' > i'$.

By the above, no good processor will r-receive any *message*(i'',j') for $i'' > i'$. Thus in step 14, for all but at most t processors j", the entries for $\mathcal{BB}_{j"}(i'',j')$ will be null. Thus, for all good processors j, $\mathcal{BB}_j(i'',j')$ is null and the lemma follows. □

LEMMA 4.5. *For all $1 < i \leq x$, each good processor j generates a value for $\mathcal{BB}[i,j]$ only after writing $\mathcal{BB}[i-1,j]$*

PROOF. For all $1 < i \leq x$, each good processor j only generates *message*(i,j) after r-receiving acks from $n-t$ processors (and at least $n-2t$ good processors) for *message*$(i-1,j)$. Since these $n-2t$ good processors sent an ack, they have not yet exited the while-loop and therefore *message*$(i-1,j)$ was r-received by them by step 11. Thus, by Lemma 4.2, it will appear in $\mathcal{BB}[i-1,j]$. Hence it was written before *message*$[(i,j)$ was generated. □

THEOREM 4.6. *If $t < n/4$, Algorithm 4 implements an x-synch in the message-passing model in time $O(x)$.*

PROOF. Correctness follows from Lemmas 4.3, 4.4, and 4.5.

All good processors will exit the while loop in $O(x)$ time. This is true since each of the reliable broadcasts by good processors take $O(1)$ time (see Lemma 3.1). Any message that is r-received by some good processor will be r-received by all good processors within 2 time steps.

In step 9 if a good processor reliably broadcasts a matrix, it will take $O(1)$ time to r-receive all entries for that matrix. Thus it takes $O(1)$ additional time for all good processors to complete this step. A similar analysis holds to show that step 9 adds

$O(1)$ time. Finally the time to complete step 13 is $O(1)$, which is the time for reliable broadcast for good processors. □

4.3. The GLOBAL-COIN algorithm

Algorithm 5 creates a candidate global coin in the asynchronous blackboard model. For each good processor p, each message that p broadcasts in the n-synch is the outcome of a fair coinflip.

ALGORITHM 5: GLOBAL-COIN for processor p

{*Perform an n-synch:* }
1: Participate in an n-synch in which each message (i, p) is the outcome of a private fair coinflip.
{*After terminating the $n - synch$:* }
2: {Exclude from V_p:} If the sum of the coins in any column j have absolute value greater than $5\sqrt{n \ln n}$, remove j from V_p.
{*Compute possible global coinflip :* }
3: Sum up the entries in the blackboard $\mathcal{BB}_p$ in columns V_p and set one's vote to the sign of the sum.

Let $\mathcal{M}$ denote the $m \times n$ matrix such that $\mathcal{M}(i, j)$ is the sum of coinflips written by processor j in the i-th iteration of MODIFIED-BRACHA by processor j. Note that we use the term "coinflips" to also refer to values set by the adversary.

Let $\mathcal{M}_p$ denote the $m \times n$ matrix such that $\mathcal{M}_p(i, j)$ is the sum of coinflips received by p in the i-th iteration of MODIFIED-BRACHA from processor $j \in V_p$ (for all $j \notin V_p$, $\mathcal{M}_p(i, j) = 0$ for all i).

LEMMA 4.7. GLOBAL-COIN *has the following properties:*

(*1*) $|\mathcal{M}_p(i, j) - \mathcal{M}(i, j)| \leq 1$ *for all processors p, for each iteration i, and for all $j \in V_p$.*
(*2*) *For any row i, there are $n - t$ entries j such that $\mathcal{M}(i, j)$ is the sum of n coinflips and $\mathcal{M}_p(i, j) = \mathcal{M}(i, j)$.*
(*3*) *With probability greater than $1 - 1/n^8$, for any iteration i and any good processor j, $\mathcal{M}(i, j) \leq 5\sqrt{n \log n} - 1$.*

PROOF. The first two properties follow directly from the properties of the blackboard model. For the last property, $\mathcal{M}(i, j)$ is the sum of up to n coinflips by a good processor. Thus, this follows as a consequence of Chernoff bounds (see Fact 1 below): $Pr(X \geq 5\sqrt{n \ln n} - 1) < Pr(X \geq 4\sqrt{n \ln n}) \leq e^{-(16n \ln n)/2n} = 1/n^8$. □

5. PROPERTIES OF COINFLIPS

5.1. Useful lemmas about the distribution of coinflips

The *deviation* of a sequence of coinflips is the absolute value of its sum. We refer to the sign of the sum as the *direction of the deviation*. Below we set $\alpha = \sqrt{2n(n - 2t)}$ and $\beta = \alpha - 2t$. We first analyze the deviations of the coinflips generated by the good processors. We use the following facts about distributions of random coinflips:

Fact 1: (Chernoff): Let X be the sum of N independent coinflips. Then for any positive a, $Pr(X \geq a) \leq e^{-a^2/2N}$.

Fact 2: ([Feller 1968], Chapter 7) Let X be the sum of N independent coinflips. Let $\Phi(a) = 1/\sqrt{2\pi} \int_{-\infty}^{a} e^{-1/2y^2} dy$. Then $Pr(X > a\sqrt{N})$ converges to $1 - \Phi(a) > (1/a - 1/a^3)(1/\sqrt{2\pi})e^{-a^2/2}$ E.g., $Pr(X > \sqrt{2N}) > 1/20$.

By Fact 2 and the symmetry of +1's and -1's:

LEMMA 5.1. *A set of at least $n(n-2t)$ good coinflips has a deviation of $\alpha = \sqrt{2n(n-2t)}$ in any specified direction with probability at least 1/20.*

LEMMA 5.2. *A set of no more than nt good coinflips has a deviation of more than $\beta/2 = \sqrt{2n(n-2t)}/2 - t$ with probability at most $e^{-(\beta/2)^2/2tn}$. If $t < n/36$, then $\beta/2 > 23n/36$ and this probability is at most $e^{-((23/36)n)^2/(2n^2(1/36))} < e^{-11}$.*

From now on, we assume $t < n/36$.

5.2. No agreement implies unusual deviation by bad processors

In this subsection, we assume that no more than t good processors have been removed from V_p for any p and assume that there is no agreement over $m \geq n$ iterations. We let G be the set of all good processors.

We show (Lemma 5.5) that this implies there is a set P' of $\Theta(n)$ good processors, such that for every $p \in P'$, there are $\Theta(m)$ iterations in which p detects large (at least $\beta/2$) deviation by bad processors.

For each iteration of MODIFIED-BRACHA, there is a particular value for the global coin flip (+1 or -1) which will result in agreement. We call this the *correct direction*. For each iteration $i \in I$ let d_i be its correct direction. For a given iteration or row, we say that a sum is *in the correct direction* if its sign agrees with the sign of the correct direction.

LEMMA 5.3. *Assume that the number of good processors in $V \setminus V_p$ is no greater than t for all processors p, and agreement is not achieved in $m \geq n$ iterations of* MODIFIED-BRACHA. *Then, with probability at least $1 - e^{-\Omega(n)}$, in $m \geq n$ iterations of* MODIFIED-BRACHA*, there are at least $0.046m$ iterations I with the following properties:*

(1) *For all $i \in I$, $d_i \sum_{j \in G} \mathcal{M}[i,j] > \alpha$; and*
(2) *For all $i \in I$, there are $0.99n - t$ good processors S_i such that for all $p \in S_i$, $d_i \sum_{j \in V \setminus V_p} \mathcal{M}[i,j] < \beta/2$.*

We denote the (bad) event that these conditions do not occur as $\mathcal{E}_1$.

PROOF. We start with condition (1). Let X be the number of rows i in which $d_i \sum_{j \in G} |\mathcal{M}[i,j]| > \alpha$. From Lemma 5.1, $E(X) \geq m/20$. Using Chernoff bounds, $Pr(X < (1-e^{-4})m/20) = e^{-me^{-8}/40}$. This implies $Pr(X < 0.049m) \leq e^{-\Omega(n)}$.

We now consider condition (2). For a processor p, let $Y_p = 1$ if $d_i \sum_{j \in V \setminus V_p} \mathcal{M}[i,j] \geq \beta/2$. Since $|V \setminus V_p| \leq t$, Lemma 5.2 shows that $Pr(Y_p = 1) < e^{-11}$. Let $Y = \sum_{p \in G} Y_p$. Then $E[Y] < (n-t)e^{-11}$.

By Markov's Inequality, $Pr(Y > 0.01n) < \frac{(n-t)e^{-11}}{0.01n} < e^{-6}$. Hence the expected number of iterations in which the event $Y > .01n$ occurs is at most me^{-6}. Let Z be the number of iterations in which the event occurs. Since each iteration is independent, we can use Chernoff bounds to bound Z: $Pr(Z \geq (1+e^{-2})me^{-6}) \leq e^{-me^{-10}/3}$. This implies $Pr(Z \geq 0.003m) = e^{-\Omega(n)}$.

By a union bound, $Pr(Z < 0.003m)$ and $X \geq 0.049m)$ is $1 - e^{-\Omega(n)}$. But if both $Z < 0.003m$ and $X \geq 0.049m$, then there are at least $X - Z > 0.046m$ iterations satisfying conditions (i) and (ii). □

The next lemma shows that if there is high deviation in the correct direction by good processors, and the deviation of the coinflips by bad processors is low, then agreement will result.

LEMMA 5.4. *Fix an iteration i of* MODIFIED-BRACHA. *If*
(1) $d_i \sum_{j \in G} \mathcal{M}[i,j] > \alpha$*;*
and if there are $2n/3$ *good processors* G_i *such that:*
(2) for every $p \in G_i$*,* $d_i \sum_{j \in V \setminus V_p} \mathcal{M}[i,j] < \beta/2$*; and*
(3) for every $p \in G_i$*,* $-d_i \sum_{j \in B \cap V_p} \mathcal{M}_p[i,j] < \beta/2$*;*
then all good processors will agree on a global coin in the correct direction, and all processors will come to agreement in the next iteration of MODIFIED-BRACHA.

PROOF. By Statement (1) of Lemma 4.7, all processors will receive all coinflips generated by good processors except at most 1 coinflip from each of as many as t good processors. Hence the adversary may cause at most a t change in deviation in the distribution of these otherwise random coinflips r-received from good processors. There is also the special case when a good processor generates a private coinflip and then is taken over by the adversary before this coinflip is written to the blackboard. This results in a possible change to the deviation of t over the course of the entire algorithm.

Fix any processor $p \in G_i$. If the deviation of the coins from good processors in $V \setminus V_p$ is less than $\beta/2$, and the deviation of the coins from bad processors which are in V_p is less than $\beta/2$, then $d_i \sum_j \mathcal{M}_p[i,j] > \alpha - \beta - 2t = 0$.

Thus, the global coin will be in the correct direction for all processors in G_i. Hence, by Lemma 3.2 (2), the next iteration of MODIFIED-BRACHA will result in Byzantine agreement. □

The next lemma gives processors a tool for singling out processors which are exhibiting unusually high deviation.

LEMMA 5.5. *Assume that:* $t < n/36$*; for each good processor* p*, the number of good processors in* $V \setminus V_p$ *is no more than* t*; and agreement is not achieved in* $m \geq n$ *iterations. Then, with probability* $1 - Pr(\mathcal{E}_1)$*, there is a set of* $0.026n$ *good processors,* P' *such that for every processor* $p \in P'$ *there is a set* I_p *of greater than* $c_1 m$ *"good" iterations such that for every iteration* $i \in I_p, |\sum_{j \in B \cap V_p} \mathcal{M}_p[i,j]| \geq \beta/2$*. (Here,* $c_1 = 0.001$*.)*

PROOF. By Lemma 5.3, with probability $1 - Pr(\mathcal{E}_1)$, there is a set I' of $0.046m$ iterations such that precondition (1) of Lemma 5.4 is satisfied for each iteration in I'. Also by Lemma 5.3, for each iteration $i \in I'$, there is a set of $0.99n - t > 2n/3$ good processors which satisfy precondition (2) of Lemma 5.4. Let S_i be that set.

Since agreement has not been reached, for every iteration $i \in I'$, there is no set of $2n/3$ processors in S_i such that precondition (3) holds for all processors in that set. Hence precondition (3) must not hold for at least $|S_i| - 2n/3 \geq (0.99n - t) - 2n/3 \geq 0.052n$ processors total (for $t < n/36$). In particular, there is a set T_i of at least $0.052n$ processors, such that for every $p \in T_i$, $-d_i \sum_{j \in B \cap V_p} \mathcal{M}_p[i,j] \geq \beta/2$.

We now use an averaging argument to show for more than $0.026n$ good processors p, $-d_i \sum_{j \in B \cap V_p} \mathcal{M}_p[i,j] \geq \beta/2$ for at least $0.026|I'|$ iterations i. There are $0.052n|I'|$ "good" processor-iteration pairs (p,i) for which $-d_i \sum_{j \in B \cap V_p} \mathcal{M}_p[i,j] \geq \beta/2$. Let x be the number of processors which appear in at least $0.026|I'| = 0.0001m$ "good" pairs. Then these x processors appear in at most $x|I'|$ "good" pairs, and the remaining $n - 0.026n$ processors each appear in fewer than $0.026n|I'|$ "good" pairs. Then the total number of "good" pairs is less than $x|I'| + n(0.026|I'|)$, which is greater than $0.052n|I'|$ only when $x > 0.026n$. The lemma follows by setting I_p to $\{i : (p,i)$ is a "good" processor-iteration pair $\}$ □

6. PROCESS-EPOCH, VARIANT 1

The previous section leaves the computational problem for each processor in P' of identifying a suitably sized submatrix each of whose columns sum to a number whose absolute value exceeds $\beta/2$. Finding such a set of columns with the requisite sums does not imply that these columns in the set are all controlled by bad processors, but it is a first step. In this section, we give a polynomial time algorithm for each processor p to measure a processor's contribution to these sums as a means of deciding whether to remove that processor from V_p.

Here we describe one method for implementing PROCESS-EPOCH, which is called by algorithm BYZANTINE-AGREEMENT. This implementation requires only polynomial time computation by each processor and results in a BYZANTINE-AGREEMENT algorithm with expected $O(n^3)$ latency. In the next section, we give an alternative implementation.

ALGORITHM 6: PROCESS-EPOCH- Variant 1

1: **if** $|\mathcal{M}_p| \geq (\beta/2)\sqrt{c_1 m/t}$ **then**
2: Let $\boldsymbol{r_p}$ be the top right singular vector of $\mathcal{M}_p$
3: For each $1 \leq i \leq n$, increase $\mathbf{score}_p(i)$ by $(\boldsymbol{r_p}[i])^2$
4: For each $1 \leq i \leq n$ remove processor i from V_p if $\mathbf{score}_p(i) \geq 1$
5: **end if**

Throughout this section, we will be using the 2-norm of vectors and matrices. The 2-norm of a vector v is $|v|_2 = \sqrt{\sum_i v_i^2}$. The 2-norm of a matrix $\mathcal{M}$ is $|\mathcal{M}|_2 = \max_{|u|_2=1} |\mathcal{M}u|_2$. We will drop the subscript 2 from all norms for notational clarity. Recall that $\beta/2 = \sqrt{2n(n-2t)}/2 - t$.

The following is a restatement of Theorem 3 from [Achlioptas and McSherry 2001].

THEOREM 6.1. *[Achlioptas and McSherry 2001] Let $\mathcal{R}$ be a random m by n matrix such that $\mathcal{R}(i,j) = r_{ij}$ where $\{r_{ij}\}$ are independent random variables and for all i,j : $r_{ij} \in [-K, K]$, $E(r_{ij}) = 0$ and $Var(r_{ij}) \leq \sigma^2$. For any $\gamma \geq 1$, $\epsilon > 0$ and $m + n \geq 20$, if*

$$K \leq \left(\frac{4\epsilon}{4+3\epsilon}\right)^3 \frac{\sigma\sqrt{m+n}}{\log^3(m+n)}$$

then

$$Pr(|\mathcal{R}| > (2+\gamma+\epsilon)\sigma\sqrt{m+n}) < (m+n)^{-\gamma^2}$$

The remaining lemmas in this section hold for any fixed epoch e.

Let $\mathcal{G}$ be the matrix $\mathcal{M}$ where all entries in columns controlled by processors in B are set to 0. Let $\mathcal{E}_2$ denote the (bad) event that some entry in $\mathcal{G}$ will have absolute value exceeding $5\sqrt{n \ln n} - 1$. From Lemma 4.7 and the union bound, $Pr(\mathcal{E}_2) < mn/n^8 = m/n^7$.

COROLLARY 6.2. *Conditioned on $\overline{\mathcal{E}_2}$, for every $\epsilon > 0$, for n sufficiently large,*

$$Pr(|\mathcal{G}| > (3+\epsilon)\sqrt{n(m+n)}) < (m+n)^{-1}.$$

PROOF. Note that $\mathcal{G}[i,j]$ are independent random variables with $\mathcal{G}[i,j] \in [-5\sqrt{n \ln n} + 1, 5\sqrt{n \ln n} - 1]$, $E(\mathcal{G}[i,j]) = 0$ and $Var(\mathcal{G}[i,j]) = \sigma^2 \leq n$. Let $\alpha = 1$ and $\epsilon > 0$ in Theorem 6.1. Then for any positive constant ϵ, for n sufficiently large, the precondition of Theorem 6.1 is satisfied and the result follows. □

For every good processor p, let $\mathcal{G}_p$ be the matrix $\mathcal{M}_p$ where all entries in columns controlled by processors not in $G \cap V_p$ are set to 0.

LEMMA 6.3. *Assume* $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n)}$*), for any fixed* $\epsilon > 0$*. Then for all* p*,* $|\mathcal{G}_p| \leq (4+\epsilon)\sqrt{n(m+n)}$*.*

PROOF. Fix a processor p and let $\mathcal{G}'_p$ be $\mathcal{M}$ where all entries in columns controlled by processors not in $G \cap V_p$ are set to 0. It is easy to see that $|\mathcal{G}'_p| \leq |\mathcal{G}|$. By Lemma 4.7, for any processor j and iteration i, $|\mathcal{M}_p(i,j) - \mathcal{M}(i,j)| \leq 1$ for all $j \in V_p$. Hence $\mathcal{G}_p = \mathcal{G}'_p + \mathcal{A}$ where all entries of $\mathcal{A}$ are integers between -1 and 1. Clearly, $|\mathcal{A}| \leq \sqrt{mn}$. We thus have $|\mathcal{G}_p| = |\mathcal{G}'_p + \mathcal{A}| \leq |\mathcal{G}'_p| + |\mathcal{A}| \leq |\mathcal{G}| + \sqrt{mn}$ and the result follows by Corollary 6.2. □

For a given processor p, let $\mathcal{B}_p$ be the matrix $\mathcal{M}_p$ where all entries in columns controlled by processors not in $B \cap V_p$ are set to 0.

For the remainder of this section, we upperbound t so that the following inequality holds: $(4+\epsilon)\sqrt{n(m+n)} < (0.183)(\beta/2)\sqrt{c_1 m/t}$, for some constant $\epsilon > 0$. When $t < n/36$, $\beta/2 > 23n/36$. We also have $m = 2n$, see proof of Lemma 6.6. Then this inequality holds for $t < 4.25 * 10^{-7} n < (0.183)^2(23/36)^2(1/3)(.002)(4+\epsilon)^{-1}n$. For the remainder of this section, we assume this bound on t.

LEMMA 6.4. *Assume* $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n)}$*. Then for any processor* p *such that* $|\mathcal{B}_p| \geq (\beta/2)\sqrt{c_1 m/t}$*, we have* $|\mathcal{G}_p| \leq 0.183|\mathcal{B}_p|$*.*

PROOF. By Lemma 6.3, $|\mathcal{G}_p| \leq (4+\epsilon)\sqrt{n(m+n)}$. Then we have:

$$\begin{aligned} |\mathcal{G}_p| &\leq (4+\epsilon)\sqrt{n(m+n)}; \\ &\leq 0.183(\beta/2)\sqrt{c_1 m/t}; \\ &\leq 0.183|\mathcal{B}_p|; \end{aligned}$$

where the second line follows by the bound on t discussed in the paragraph preceding this lemma. □

LEMMA 6.5. *Let* P' *be the set of processors as defined in Lemma 5.5 and assume* $\overline{\mathcal{E}_1}$*. For any processor* $p \in P'$*,*

$$|\mathcal{B}_p| \geq (\beta/2)\sqrt{c_1 m/t}.$$

Also, if $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n)}$*, then* $|\mathcal{G}_p| \leq (0.183)|\mathcal{B}_p|$*.*

PROOF. Let t_p be the number of bad processors in V_p. Let x be a length t_p unit vector, where all entries equal $1/\sqrt{t_p}$. Consider the vector $\boldsymbol{y} = \mathcal{B}_p \boldsymbol{x}$. Note that for at least $c_1 m$ entries of $\boldsymbol{y}$, the square of the value of that entry is at least $(\beta/2)^2/t_p$. Hence $|\boldsymbol{y}| \geq (\beta/2)\sqrt{c_1 m/t_p} \geq (\beta/2)\sqrt{c_1 m/t}$. The second inequality follows from Lemma 6.4. □

For simplicity of analysis, we assume that the columns of $\mathcal{M}_p$ are arranged so that the columns for the t bad processors are to the left of the columns for the $n-t$ good processors. We note that this rearrangement is equivalent to multiplying $\mathcal{M}_p$ by a permutation matrix and so will not effect the singular values of $\mathcal{M}_p$.

Now let $\boldsymbol{\ell_p}$ and $\boldsymbol{r_p}$ be the top left and right singular vectors of $\mathcal{M}_p$. Note that by definition, $|\mathcal{M}_p| = \boldsymbol{\ell_p}^T \mathcal{M}_p \boldsymbol{r_p}$.

Our analysis will focus on $\boldsymbol{r_p}$. Let $\boldsymbol{b_p}$ be defined such that for all $1 \leq i \leq t$, $\boldsymbol{b_p}[i] = \boldsymbol{r}[i]$ and all other entries of $\boldsymbol{b_p}$ are 0. Similarly, define $\boldsymbol{g_p}$ such that for all $t+1 \leq i \leq n$, $\boldsymbol{g_p}[i] = \boldsymbol{r}[i]$ and all other entries of $\boldsymbol{g_p}$ are 0. Note that by construction, $\boldsymbol{r_p} = \boldsymbol{b_p} + \boldsymbol{g_p}$.

LEMMA 6.6. *Assume* $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n)})$*. Then for every processor* p *such that* $|\mathcal{M}_p| \geq (\beta/2)\sqrt{c_1 m/t}$*,* $|\boldsymbol{g_p}|^2 < |\boldsymbol{b_p}|^2/2$*.*

PROOF. Assume by way of contradiction that $|\boldsymbol{g_p}|^2 \geq |\boldsymbol{b_p}|^2/2$. Note that $|\boldsymbol{g_p}|^2+|\boldsymbol{b_p}|^2 = |\boldsymbol{r_p}|^2 = 1$. Thus, we have

$$\begin{aligned} 1 &= |\boldsymbol{g_p}|^2 + |\boldsymbol{b_p}|^2 \\ &\geq |\boldsymbol{b_p}|^2/2 + |\boldsymbol{b_p}|^2 \\ &= 3/2|\boldsymbol{b_p}|^2 \end{aligned}$$

This implies that $|\boldsymbol{b_p}|^2 \leq 2/3$ or $|\boldsymbol{b_p}| \leq \sqrt{2/3}$. We further note that $|\boldsymbol{g_p}|^2 \leq 1$, so $|\boldsymbol{g_p}| \leq 1$.

Now $\mathcal{M}_p \boldsymbol{r_p} = \mathcal{B}_p \boldsymbol{b_p} + \mathcal{G}_p \boldsymbol{g_p}$. Hence $|\mathcal{M}_p \boldsymbol{r_p}| \leq |\mathcal{B}_p||\boldsymbol{b_p}| + |\mathcal{G}_p||\boldsymbol{g_p}|$.

Putting this together, we have:

$$\begin{aligned} |\mathcal{B}_p| &\leq |\mathcal{M}_p| \\ &= \boldsymbol{\ell_p}^T(\mathcal{M}_p \boldsymbol{r_p}) \\ &\leq |\boldsymbol{\ell_p}||\mathcal{M}_p \boldsymbol{r_p}| \\ &\leq |\mathcal{B}_p||\boldsymbol{b_p}| + |\mathcal{G}_p||\boldsymbol{g_p}| \\ &\leq |\mathcal{B}_p|(|\boldsymbol{b_p}| + 0.183|\boldsymbol{g_p}|) \quad \text{by Lemma 6.4} \\ &\leq |\mathcal{B}_p|(\sqrt{2/3} + 0.183) \\ &< |\mathcal{B}_p| \end{aligned}$$

which is clearly a contradiction. In the above inequalities, the third line follows by the Cauchy-Schwartz inequality. □

LEMMA 6.7. *With probability at least 1/2, Algorithm 1, using* PROCESS-EPOCH *Variant 1, will terminate successfully in* $116t$ *epochs, each consisting of* $m = 2n$ *iterations, with resilience* $t < 4.25 \times 10^{-7}n$. *When the algorithm terminates, all processors will achieve Byzantine agreement. The algorithm is Las Vegas with no more than* $232tm < n^2$ *expected iterations of* MODIFIED-BRACHA *for a total of* $O(n^3)$ *rounds of communication and polynomial time computation.*

PROOF. By Corollary 6.2, in a fixed epoch, with probability at least $1 - 1/(m+n) - Pr(\mathcal{E}_2)$, $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n)}$. As $Pr(\mathcal{E}_2) < m/n^7$ (see discussion before Corollary 6.2), this condition holds for an epoch with probability at least $1 - 1/(m+n) - m/n^7$.

As long as this condition holds for each epoch, we claim that no more than t good processors are removed from V_p for any processor p. First, observe that each processor v is removed from V_p when $\mathbf{score}_p(v) \geq 1$. Each epoch can add no more than 1 to this total for any processor. Hence, the maximum $\mathbf{score}_p$ accrued by a processor before its removal is less than 2.

Assume by way of contradiction that more than t good processors have been removed from V_p. Then $\sum_{i \in G} \mathbf{score}_p(i) > t$. In any epoch where processor p adds to the score values, it must be the case that $|\mathcal{M}_p| \geq (\beta/2)\sqrt{c_1 m/t}$. Then Lemma 6.6 implies $|\boldsymbol{b_p}|^2 \geq 2|\boldsymbol{g_p}|^2$, Thus, by our assumption, $\sum_{i \in B} \mathbf{score}_p(i) > 2t$. Since there are no more than t bad processors, this implies that for at least one bad processor i, $\mathbf{score}_p(v) \geq 2$, giving a contradiction.

We next show that it is likely that all bad processors will be removed from each V_p for all $p \in G$ in $116t$ epochs. By Lemma 6.5, if event $\mathcal{E}_1$ does not occur, then in every epoch in which there is no agreement, there are $0.026n$ processors in the set P' such that $|M_p| \geq |B_p| \geq \beta/2\sqrt{c_1 m/t}$. The value of $\sum_i \mathbf{score}_p(i)$ must increase by 1 for each of these processors in a set P'.

Let T be the number of such epochs before all bad processors are removed from all V_p, assuming no agreement occurs before then. Then with probability at least $1 - TPr(\mathcal{E}_1)$,

$\sum_p \sum_i \mathbf{score}_p(i) \geq 0.026nT$. As $2/3$ of this score is attributable to processors in B, and there are no more than t processors in B, which each accrue a score of less than 2, we know that $(2/3)(.026nT) < n(2t)$. Solving for T, we find $T < 116t$.

Then with probability at least $1 - T(1/(m+n) + Pr(\mathcal{E}_2) + Pr(\mathcal{E}_1)) > 1 - 116t(1/(m+n) + m/n^7 + 1/e^{\Omega(n)} > 2/3$, the entire algorithm will successfully run for less than $116t$ epochs until all bad processors are removed. Once there are no more bad processors, it will succeed with constant probability in any iteration and over the next epoch it is very likely to succeed once. Thus total probability of success within $116t$ epochs is at least $1/2$. If the algorithm fails, it will repeat until Byzantine agreement is decided, making the algorithm Las Vegas with an expected $232t$ epochs.

As each epoch contains $O(n)$ iterations of MODIFIED-BRACHA; each execution of MODIFIED-BRACHA contains one GLOBAL-COIN which in turn contains $O(n)$ rounds of communication, the total latency is $O(n^3)$ expected time.

We can analyze computation time per processor as follows. In the GLOBAL-COIN algorithm, the n-synch costs $O(n^4)$ computation per processor. In particular, the reliable broadcast of a message requires each processor to receive $O(n)$ messages. Since $O(n)$ n by n matrices are being reliably broadcast during the Spread and Final Update parts of the n-synch, each processor receives $O(n^2)$ messages, each of size $O(n^2)$. After each epoch the computation of the singular value decomposition requires $O(n^3)$.

Since there are $O(n)$ expected epochs, each with $O(n)$ calls to GLOBAL-COIN, the expected computation cost per processor is $O(n^6)$. □

This concludes the proof of Theorem 1.1.

7. PROCESS-EPOCH- VARIANT 2

Fix an epoch e. During that epoch, when and if p determines there is a set $S_p \subseteq V_p$ of no more than t processors and a set of $c_1 m$ iterations I_p such that for each iteration $i \in I_p$, $|\sum_{j \in S_p} \mathcal{M}_p(i,j)| \geq \beta/2$, p also determines for each processor $v \in S_p$ the individual *epoch deviation* of the processor v. For each $i \in I_p$ let $\mathbf{sign}_p(i)$ be the sign of $\sum_{j \in S_p} \mathcal{M}_p(i,j)$. Here we assume each processor will find a set S_p if it exists and this may take exponential computation time. We also assume $t < n/400$.

ALGORITHM 7: PROCESS-EPOCH- Variant 2

1: **if** p finds a set S_p of processors of size at most t and a set of $c_1 m$ iterations I_p in epoch e such that for every iteration $i \in I_p$, $|\sum_{j \in S_p} \mathcal{M}_p(i,j)| \geq \beta/2$ **then**
2: **for** each $v \in S_p$ **do**
3: $\mathbf{score}_p(v) \leftarrow \mathbf{score}_p(v) + \sum_{i \in I_p} \mathcal{M}_p(i,v)\mathbf{sign}_p(i)$.
4: **if** $\mathbf{score}_p(v) \geq 2(5\sqrt{n \ln n})c_1 m$ **then** remove v from V_p
5: **end for**
6: **end if**

7.1. Analysis

Recall that B is the set of bad processors and G is the set of good processors. For a good processor p, a set of processors S, and an epoch e, let $\mathbf{score}_p(S,e)$ be the sum over all processors $q \in S$ of the amount added to $\mathbf{score}_p(q)$ in epoch e.

LEMMA 7.1. *Assume Algorithm 1 using* PROCESS-EPOCH- *Variant 2 runs for at most a polynomial number of epochs and* $t < n/400$ *and* $m = 1000n$. *Then with probability* $1 = e^{-\Omega(n)}$, *for any good processor* p *and any epoch* e, $\mathbf{score}_p(G,e) \leq (\beta/5)c_1 m$.

PROOF. Assume that in some epoch e, p finds a set of a most t processors S_p and a set of $c_1 m$ iterations I_p such that for every iteration $i \in I_p$, $|\sum_{j \in S_p} \mathcal{M}_p(i,j)| \geq \beta/2$. We note that p only possibly changes $\mathbf{score}_p(q)$ for processors $q \in S_p$. Therefore we would like to bound the amount added for all $q \in G \cap S_p$.

Let $G'_p = G \cap S_p$ and let $X = \sum_{i \in I_p, j \in G'_p} \mathcal{M}(i,j)\mathbf{sign}_p(i)$. By Lemma 4.7 (1), $|\mathcal{M}_p(i,j) - \mathcal{M}(i,j)| \leq 1$ for all processors p, for each processor j and iteration i. Recall that $\beta = \sqrt{2n(n-2t)} - 2t$. Thus, if $X \leq (\beta/6)c_1 m$, this implies that $\mathbf{score}_p(G,e) = \mathbf{score}_p(G'_p,e) \leq X + tc_1 m \leq (\beta/6 + t)c_1 m \leq (\beta/5)c_1 m$ when $t < \beta/30$ which is true since $t \leq n/400$. Further note that the probability that $X \geq (\beta/6)c_1 m$ is maximized when all processors in G'_p generate all n coinflips for each iteration in I_p. Thus, we will pessimistically assume this is the case throughout the proof.

We now fix a set G'_p, a set I_p and a mapping $\mathbf{sign}$ from iterations in I_p to $\{-1, 1\}$. Since X is the sum of independent trials, by Fact 1, and the fact that $\beta^2 \geq 1.9n^2$ (since $t \leq n/400$), we have:

$$\begin{aligned} Pr(X \geq (\beta/6)c_1 m) &\leq e^{-(c_1 m\beta/6)^2/2(|G'_p|c_1 mn)} \\ &\leq e^{-c_1 m\beta^2/(n72|G'_p|)} \\ &\leq e^{-c_1 m\beta^2/(72nt} \\ &\leq e^{-0.026c_1 mn/t} \end{aligned}$$

There are $\binom{m}{c_1 m} \leq (me/c_1 m)^{c_1 m}$ ways to pick the iterations I_p; at most $\sum_{i=1}^{t}\binom{n}{i} \leq 2^n$ ways to pick the set G'_p; and $2^{c_1 m}$ ways to pick the mapping d. Let ξ be the event that $X \geq (\beta/6)(c_1 m)$ for *any* sets G'_p and I_p, and mapping $\mathbf{sign}$. Then by a union bound, we have the following.

$$\begin{aligned} Pr(\xi) &\leq (me/c_1 m)^{c_1 m} 2^n 2^{c_1 m} e^{-c_1 m\beta^2/(72nt)} \\ &\leq e^{8c_1 m} e^{c_1 m} e^{c_1 m} e^{-0.026c_1 mn/t} \\ &\leq e^{10c_1 m - (0.026c_1 mn/t)} \\ &\leq e^{-\Omega(n)} \qquad \text{since } n/t \geq 400 \end{aligned}$$

Another union bound over all good processors p and the polynomial number of epochs e establishes the result. □

LEMMA 7.2. *With probability $1 - e^{-\Omega(n)}$, for all processors p, the number of good processors removed from V_p in T epochs is no more than t.*

PROOF. Suppose a processor p detects a set S_p as defined in Algorithm 1 in some epoch e, for iterations I_p. From Lemma 7.1, with probability $1 - e^{-\Omega(n)}$, $\mathbf{score}_p(G \cap S_p, e) < (\beta/5)c_1 m$. Since the sum of the $\mathbf{score}_p$ values assigned in the epoch is at least $\beta/2$ for each of the $c_1 m$ iterations, this implies that $\mathbf{score}_p(B \cap S_p) > (3\beta/10)c_1 m$. Hence, the total increase in score of bad processors in V_p is greater than 3/2 the total increase in score of good processors in V_p in each epoch.

Fix some epoch. Then we show $|G \setminus V_p| \leq t$. Suppose to the contrary, $|G \setminus V_p| > t$. Then $\sum_{v \in G \setminus V_p} \mathbf{score}_p(v) \geq 2(5\sqrt{n \ln n}tc_1 m$. This implies $\sum_{v \in B} \mathbf{score}_p(v) > 3(5\sqrt{n \ln n})tc_1 m$.

Each processor v is removed from V_p when $\mathbf{score}_p(v) \geq 2(5\sqrt{n \ln n})c_1 m$. Each epoch, including a processor's last before its removal, adds no more than $(5\sqrt{n \ln n})c_1 m$ to $\mathbf{score}_p(v)$. For any processor v, $\mathbf{score}_p(v) < 3(5\sqrt{n \ln n})c_1 m$. This together with the fact that there are at most t bad processors gives a contradiction □

LEMMA 7.3. *With probability* $1 - O(1/n^5)$*, Algorithm 1 using* PROCESS-EPOCH *Variant 2 will terminate in* $O(\sqrt{n \log n})$ *epochs (*$O(n^{2.5}\sqrt{\log n})$ *communication time. The algorithm is Las Vegas with expected communication time that is* $O(n^{2.5}\sqrt{\log n})$.

PROOF. With probability $1 - n[Pr(\mathcal{E}_2) + Pr(\mathcal{E}_1)]$, neither $\mathcal{E}_2$ nor $\mathcal{E}_1$ occur in the first n epochs of the algorithm. Assume they do not occur. Let X be the sum over all processors p and q of the value $\mathbf{score}_p(q)$. By the argument in the proof of Lemma 7.2, for any processors p and q, $\mathbf{score}_p(q) \leq 3(5\sqrt{n \ln n})c_1 m$. Summing over all n^2 pairs of processors, we see that $X \leq n^2(15\sqrt{n \ln n})c_1 m$. By Lemma 5.5, in each epoch in which the algorithm does not terminate, the value of X must increase by at least $(0.026n)(\beta/2)c_1 m$. Let T be the number of epochs in which the algorithm does not terminate. Thus, $X \geq T(0.026n)(\beta/2)c_1 m$. Since $X \leq n^2(15\sqrt{n \ln n})c_1 m$, this implies $T \leq n^2 15\sqrt{n \ln n}/(0.026n)(\beta/2) = O(\sqrt{n \log n})$. We conclude that with probability $1 - n(e^{-\Omega(n)} + m/n^7) = 1 - O(1/n^5)$, the Algorithm terminates within $O(\sqrt{n \log n})$ epochs.

With very small probability, the entire algorithm will repeat until Byzantine agreement is decided, making the algorithm Las Vegas with $O(n^{2.5}\sqrt{\log n})$ communication time. □

This concludes the proof of Theorem 1.2.

8. CONCLUSION AND FUTURE WORK

We have described two algorithms to solve Byzantine agreement in the classic asynchronous message-passing model in polynomial communication time. Our algorithm works in the asynchronous message-passing model, when an adaptive and full-information adversary controls a constant fraction of the processors. We demonstrated that cryptographic assumptions or private channels are not necessary to solve this long-standing fundamental problem. This may perhaps open the door to the solution of other problems in a full-information model.

Our algorithm is designed so that in order to thwart it, corrupted nodes must engage in statistically deviant behavior that is detectable by individual nodes. This suggests a new paradigm for randomized distributed computing: the design of algorithms which force attackers into behavior which a good processor might possibly engage in but is statistically unlikely, and which is detectable in polynomial time.

Our result leaves much room for improvement, in terms of the resilience and expected communication time. Can the resilience be increased to the optimal bound of $t < n/3$? Can we decrease the expected communication time to $\tilde{O}(n^{2.5})$ with polynomial time computation? An intriguing open question is whether the expected communication time can be brought down to the known lower bound of $\tilde{\Omega}(n)$ in the shared memory consensus problem, or whether Byzantine agreement is intrinsically harder than consensus, in terms of time or step complexity.

REFERENCES

Dimitris Achlioptas and Frank McSherry. 2001. Fast computation of low rank matrix approximations. In *Proceedings of the thirty-third annual ACM Symposium on Theory of Computing*. ACM, 611–618.

Noga Alon, Michael Krivelevich, and Benny Sudakov. 1998. Finding a large hidden clique in a random graph. In *Proceedings of the ninth annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 594–598.

James Aspnes. 1998. Lower Bounds for Distributed Coin-Flipping and Randomized Consensus. *Journal of the Association for Computing Machinery* 45(3) (May 1998), 415–450.

J. Aspnes. 2003. Randomized Protocols for Asynchronous Consensus. *Distributed Computing* 16 (2003), 165–175.

James Aspnes and Maurice Herlihy. 1990. Fast Randomized Consensus Using Shared Memory. *Journal of Algorithms* 11, 3 (1990), 441–461.

Hagit Attiya and Keren Censor. 2008. Lower bounds for randomized consensus under a weak adversary. In *PODC*. 315–324.

Hagit Attiya and Jennifer Welch. 2004. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience.

Yossi Azar, Amos Fiat, Anna Karlin, Frank McSherry, and Jared Saia. 2001. Spectral Analysis of Data. In *Symposium on Theory of Computing (STOC)*. ACM, 619–626.

M. Ben-Or. 1983. Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM Symposium on Principles of Distributed Computing*. ACM New York, NY, USA, 27–30.

Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-optimal interactive consistency in constant time. *Distributed Computing* 16, 4 (2003), 249–262.

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

Gabriel Bracha. 1984. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *PODC*. 154–162.

Gabriel Bracha. 1987a. Asynchronous Byzantine Agreement Protocols. *Journal of Information and Computation* 75, 2 (1987), 130–143.

Gabriel Bracha. 1987b. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.

R. Canetti and T. Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*.

Soumen Chakrabarti, Byron E Dom, S Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, David Gibson, and Jon Kleinberg. 1999. Mining the Web's Link Structure. *Computer* 32, 8 (1999), 60–67.

B. Chor and C. Dwork. 1989. Randomization in Byzantine agreement. *Advances in Computing Research* 5 (1989), 443–498.

Paul Feldman and Silvio Micali. 1985. Byzantine Agreement in Constant Expected Time (and Trusting No One). In *FOCS*. 267–276.

W. Feller. 1968. *An Introduction to Probability Theory, Vol. 1* (third ed.). Wiley, New York, NY.

M.J. Fischer, N.A. Lynch, and M.S. Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

Ramanthan Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. 2004. Propagation of trust and distrust. In *Proceedings of the 13th International Conference on World Wide Web*. ACM, 403–412.

Mark Jerrum. 1992. Large cliques elude the Metropolis process. *Random Structures & Algorithms* 3, 4 (1992), 347–359.

Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. 2003. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*. ACM, 640–651.

Bruce Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. 2009. Scalable Algorithms for Byzantine Agreement and Leader Election with Full Information. *ACM Transactions on Algorithms(TALG)* (2009).

A. Karlin and A. Yao. 1984. Probabilistic lower bounds for byzantine agreement and clock synchronization. (1984). Unpublished manuscript.

Valerie King and Jared Saia. 2013. Byzantine agreement in Polynomial Expected Time. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. ACM, 401–410.

Valerie King and Jared Saia. 2014. Faster Agreement via a Spectral Method for Detecting Malicious Behavior. In *Proceedings of the ACM-SIAM Symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics.

Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

Luděk Kučera. 1995. Expected complexity of graph partitioning problems. *Discrete Applied Mathematics* 57, 2 (1995), 193–212.

L. Lamport, R. Shostak, and M. Pease. 1982. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 401.

A. Lewko. 2011. The contest between simplicity and efficiency in asynchronous byzantine agreement. *Distributed Computing* (2011), 348–362.

Nancy Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann.

Gil Neiger and Sam Toueg. 1990. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms* 11, 3 (1990), 374–419.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

Li Xiong and Ling Liu. 2004. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *Knowledge and Data Engineering, IEEE Transactions on* 16, 7 (2004), 843–857.

Hui Zhang, Ashish Goel, Ramesh Govindan, Kahn Mason, and Benjamin Van Roy. 2004. Making eigenvector-based reputation systems robust to collusion. In *Algorithms and Models for the Web-Graph*. Springer, 92–104.