# Faster Agreement via a Spectral Method for Detecting Malicious Behavior

Valerie King * Jared Saia †

**Abstract**

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 amount of communication. In 2013, the algorithm was improved to expected polynomial communication time, but still an exponential amount of computation per individual processor was required. In this paper, we improve that result to require both expected polynomial computation and communication time.

We use a novel technique for detecting malicious behavior via spectral analysis. In particular, our algorithm uses coin flips from individual processors to repeatedly try to generate a fair global coin. The corrupted processors can bias this global coin by generating biased individual coin flips. However, we can detect which processors generate biased coin flips by analyzing the top right singular vector of a matrix containing the sums of coin flips generated by each processor. Entries in this singular vector with high absolute value correspond to processors that are trying to bias the global coin, and this information can be used to blacklist malicious processors.

## 1 Introduction

Random bits are used in computing to break symmetry, ensure load-balancing, find a representative sample, maximize utility, and foil an adversary. Unfortunately, randomness is difficult to guarantee, especially in a decentralized model where not all agents are guaranteed to be reliable. What happens if a hidden cabal generates bits that are not truly random? Can we detect and neutralize such behavior?

In this paper, we address this question in the context of a classic problem in distributed computing: Byzantine agreement. In the Byzantine agreement problem, $n$ agents, each with a private input, must agree on a single common output that is equal to some agent's input. Randomization is provably necessary and sufficient to solve this problem, but past randomized algorithms required expected exponential time, in the model we consider.

**Our model:** We consider Byzantine agreement in the challenging classic asynchronous model. There is a bound $t$ on the total number of processors that the adversary can take over. The adversary is *adaptive*: it can take over processors at any point during the protocol, up to the point of taking over $t$ processors.[1] 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" [3]. The major constraint on the adversary is that it cannot predict future coinflips, and 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.

*Communication Time* in this model is defined to be the maximum length of any chain of messages (see [11, 3]), and sending a message over the network is counted as taking 1 unit of communication time. In addition, we consider *computation time* by individual processors, which is measured in the usual way.

**Our Result:** In 2013 [17], the authors gave the first algorithm in this model with expected polynomial communication time. However, this algorithm required exponential computation per processor. We improve that result to require expected polynomial computation

*Department of Computer Science, University of Victoria. This research was partially supported by an NSERC grant and PIMS; email: val@uvic.ca.

†Department of Computer Science, University of New Mexico. This research was partially supported by NSF CAREER Award 0644058 and NSF CCR-0313160; email: saia@cs.unm.edu.

[1] This is in contrast to a non-adaptive adversary that chooses the $t$ processors to take over at the beginning of the algorithm.

and communication. Specifically, our main result is as follows.

THEOREM 1.1. *Let $n$ be the number of processors, then there is a $t = \Theta(n)$ such that the following holds in the asynchronous message passing model with an adaptive, full-information adversary that controls up to $t$ processors. Byzantine Agreement can be solved in expected $O(n^3)$ communication time, expected polynomial computation time per processor, and expected polynomial bits of communication.*

**Neutralizer-Detector Game (simplified)** We can describe the computational problem we solve here, by a novel game between a neutralizer N and and detector D. We sketch a simplified version here and give a more exact description in Section 5. An improvement to the exact version would result in a speed-up to our Byzantine agreement protocol.

The game starts with an $m \times n$ matrix, where $m = \Theta(n)$, and proceeds in epochs on $m \times n'$ matrices, with $n'$ monotonically decreasing. Let $c < 1$ be a fixed constant. In each epoch, N must pick the values in its columns so that the sum of each row of the matrix is *neutralized*, i.e., the absolute value of each sum of each row of the matrix is less than $cn$. Let $t$ be a small constant fraction of $n$. Each epoch consists of the following phases.

Phase 1: N may claim columns, provided that the total number of columns claimed during the game is less than $t$.
Phase 2: All entries in unclaimed columns are independently set to the sum of $n$ independent fair coinflips of value +1 and -1.
Phase 3: N sets the values in its columns.
Phase 4: D may remove columns, provided that the total number of columns removed during the game is no greater than $2t$.

The game ends when N fails to neutralize a row. We show that D has a strategy with computational cost of $O(n^3)$ time per epoch, and that this strategy ends the game in an expected $O(n)$ epochs.

Note that the game may be viewed as being played over a weighted bipartite graph with node set $(R, C)$. The nodes in $R$ correspond to the rows in the matrix, and the nodes in $C$ correspond to the columns in the matrix. The weights on edges incident to unclaimed nodes in $C$ are set randomly and the weights on other edges are fixed by the adversary (N). The algorithm (D) must find a small superset of the claimed nodes. Our solution involves identifying a subgraph where the weights are higher than expected. Thus the problem is related to an iterated version of finding a weighted planted bipartite subgraph. See Section 2 for details.

**1.1 Technical Overview** In our algorithm, each processor $p$ maintains a set $V_p$ of processors that $p$ accepts coinflips from. Initially $V_p$ contains all $n$ processors. The algorithm proceeds in epochs, where each epoch is composed of $m$ iterations of a variant of Ben-Or's algorithm. In each iteration, each processor broadcasts $n$ coinflips, where heads is +1 and tails is -1. At the end of every epoch in which agreement does not occur, each processor $p$ creates a $m \times |V_p|$ matrix, $\mathcal{M}_p$, where $\mathcal{M}_p(i, j)$ is the sum of the coinflips received in iteration $i$ from processor $j$. This matrix will be used to detect bad processors, as described in Section 1.1.1. To show how this is done, we first must describe two important properties of the matrix.

- **Property 1:** For any good processor $q$, let $s(i, q)$ be the sum of coinflips broadcast by $q$ in iteration $i$. Then for any processor $p$, such that $q \in V_p$, $|\mathcal{M}_p(i, I_q) - s(i, q)| \leq 3$, where $I_q$ is the index associated with processor $q$.
- **Property 2:** There exists a set of $\Theta(n)$ good processors $P'$, such that for every $p \in P'$, the following holds for a set, $R$, $|R| = \Theta(m)$, of the rows of $\mathcal{M}_p$. For all $i \in R$, $|\sum_{j \in I_B} \mathcal{M}_p(i, j)| \geq kn$, where $k$ is a fixed constant, and $I_B$ is the set of indices associated with bad nodes.

Property 1 derives from our coin flipping algorithm, GLOBAL-COIN, from Section 3.2; see also Lemma C.5. This algorithm has been modified slightly from the algorithm in [17] so that each processor now performs only polynomial computation.

Property 2 holds via the following argument. In each iteration, the sign ("direction") of the sum of the individual coinflips is used to generate a global coinflip for our variant of Ben-Or's algorithm (Algorithm 4). Ben-Or's algorithm has the property that, in each iteration, there is a fixed direction such that, if the global coinflip is in this direction, all processors reach agreement in the next iteration. We call this the "correct" direction. In any iteration, with constant probability, the absolute value of the sum of the coinflips of all good processors is more than $2kn$ and the sum is in the correct direction. When this event occurs in an iteration, we say that the iteration is "good".

In any good iteration, the adversary must decrease the absolute value of the sum by $2kn$ in order to prevent agreement from being reached. At most $kn$ of this decrease can occur via the asynchronous scheduler. The remaining $kn$ of the decrease must occur via strategic

setting of the coinflips of the bad processors. Intuitively, the set $R$ in Property 2 is the set of good iterations for processor $p$. See Lemma 4.1 for a detailed argument.

Properties 1 and 2 were shown to hold in [17]. Our main contribution is to describe a new polynomial time algorithm PROCESS-EPOCH that uses these properties to detect bad processors from the accumulated coinflips. We next sketch this algorithm.

**1.1.1 Detecting Bad Processors** Imagine permuting the columns of $\mathcal{M}_p$ so that $\mathcal{M}_p = [\mathcal{B}_p \mathcal{G}_p]$, where $\mathcal{B}_p$ is a $m \times |B \cap V_p|$ matrix consisting of the columns in $\mathcal{M}_p$ corresponding to the bad processors, and $\mathcal{G}_p$ is a $m \times |G \cap V_p|$ matrix consisting of the columns corresponding to the good processors. From Property 2, we can prove that for all $p \in P'$, the 2-norm of the matrix $\mathcal{B}_p$ will be at least $c_1 n$ for some constant $c_1$ (Lemma 6.3). By a standard result on the 2-norm of a matrix of random variables [1], and by Property 1, we can show that, w.h.p., the 2-norm of the matrix $\mathcal{G}_p$ will be at most $c_2 n$ for a constant $c_2 < c_1$ (see Corollary 6.1 and Lemma 6.1 for details). The gap between $c_2$ and $c_1$ can be made arbitrarily large as the ratio $t/n$ decreases (Lemma 6.2).

Our technique for detecting the bad processors relies on this gap between $c_1$ and $c_2$. Specifically, let $\boldsymbol{r_p}$ be the top right singular vector of $\mathcal{M}_p$. We show that if the gap between $c_1$ and $c_2$ is sufficiently large, the values $\boldsymbol{r_p^2}[i]$ for $i$ associated with bad processors, will tend to be larger than the values $\boldsymbol{r_p^2}[j]$ for $j$ associated with good processors. In particular, we can ensure that $\sum_{i \in I_G} \boldsymbol{r_p^2}[i] < (1/2) \sum_{i \in I_B} \boldsymbol{r_p^2}[i]$, where $I_G$ is the set of indices whose columns map to good processors, and $I_B$ is the set of indices whose columns map to bad processors.

The algorithm that each processor $p$ uses to detect bad processors then is simple. Initially, $p$ sets $\mathbf{cumdev}_p(q)$ to 0 for all processors $q$. Then at the end of each epoch, $p$ computes a matrix $\mathcal{M}_p$ for that epoch as described above. If the 2-norm of $\mathcal{M}_p$ is at least $c_1 n$, then, for each processor $q$, $p$ increases $\mathbf{cumdev}_p(q)$ by $\boldsymbol{r_p^2}[q]$, where $\boldsymbol{r_p}$ is the top right singular vector of $\mathcal{M}_p$. For all processors $q$ such that $\mathbf{cumdev}_p(q)$ now exceeds 1, $p$ removes $q$ from $V_p$.

We show that, for any processor $p$, w.h.p., no more than $t$ good processors are ever removed from $V_p$ (see proof of Lemma 6.5). Once it is the case that for every good processor $p$, all bad processors are removed from $V_p$, and no more than $t$ good processors are removed from $V_p$, agreement is reached within an expected constant number of iterations. We can make the entire algorithm Las Vegas by ensuring that for every processor $p$, all $\mathbf{cumdev}_p$ values are reset to 0 and $V_p$ is reset to all $n$ processors, in the unlikely event that more than $2t$ processors are added to $p$'s blacklist.

**Paper Organization:** The rest of this paper is organized as follows. In Section 2, we discuss related work. In Sections 3.1, we present our modified version of Ben-Or's algorithm, MODIFIED-BEN-OR, which calls upon a coinflip algorithm we call GLOBAL-COIN. The GLOBAL-COIN procedure is presented in Section 3.2. In Section 4, we analyze properties of the coinflips generated and broadcast during the multiple calls to GLOBAL-COIN in an epoch. In Section 5 we describe in more detail two versions of the Neutralizer-Detector game, such that a winning strategy for D gives an algorithm for the processors in Byzantine agreement. In Section 6, we describe PROCESS-EPOCH and analyze the spectral properties of the matrices $\mathcal{M}_p$ described above. Section 7 discusses future directions and open problems.

Throughout this paper, we will use the phrase with high probability (w.h.p.) to mean with probability $1 - 1/n^c$ for any fixed constant $c$.

## 2 Related work

**Spectral Methods:** Spectral methods have been used frequently to identify trustworthy and untrustworthy agents in decentralized systems. Perhaps one of the most prominent applications is identifying trustworthy web pages. The PageRank algorithm, of Page et al. [22] (which was inspired by the spectral-based Clever algorithm of Kleinberg [18, 10]) 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 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 [24, 4].

The idea of PageRank is the basis of the eigentrust algorithm [15] (see also [23, 13, 24]). 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 singular vector are not trustworthy. Intuitively, this is because in our algorithm, good processors have random coinflips,

and so over time, the columns associated with these processors will have little "structure", which translates to a small absolute value in the singular vector.

Our neutralizer detector game shares some similarities with the hidden clique detection problem. In this problem, proposed independently by Jerrum [14] and Kucera [19], 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 [2] described a spectral algorithm that can find a clique, w.h.p., when $k = \Omega(\sqrt{n})$ [2]. Roughly, this algorithm 1) finds the second eigenvector of the adjacency matrix of $G$; 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 $G$ that have at least $3k/4$ neighbors in $W$.

**Byzantine agreement:** The number of papers published on Byzantine agreement numbers well into the tens of thousands. We refer the reader to [3, 21] 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. We note that the full-information assumption makes the problem challenging. 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. [9]).

The Byzantine agreement problem was introduced over 30 years ago by Lamport, Shostak and Pease [20]. 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 [12].

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 described above. His algorithm consists of multiple rounds in which each good processor tosses 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 $t < n/3$ in 1984 by Bracha [5]. The communication time remained exponential. This resilience is the best possible [16].

In 2013, King and Saia gave the first algorithm for the classic asynchronous model running in expected polynomial communication time. Their algorithm required $O(n^{2.5})$ expected communication time and tolerated $t < n/500$. Unfortunately, the computation time was exponential. In this paper, we achieve expected polynomial communication and computation time. However, there is a cost. Our expected communication time increases to $O(n^3)$ and our resilience decreases to $t < .000028n$.

## 3 The BYZANTINE-AGREEMENT Algorithm

The main algorithm, BYZANTINE-AGREEMENT uses two procedures. The first, MODIFIED-BEN-OR, is a modified version of Ben-Or's 1983 exponential expected time Byzantine agreement. This achieves agreement in constant expected time if there are no bad processors. If there are bad processors then MODIFIED-BEN-OR is rerun until agreement is achieved.

Let $V$ be the set of all processors. After each $m = O(n)$ iterations of MODIFIED-BEN-OR (an *epoch*), each processor $p$ uses the second procedure, PROCESS-EPOCH, to analyze data from that epoch to assign "badness" in the form of $cumdev_p(v)$ to processors $v$. Using these values, $p$ maintains a subset of the remaining possibly good processors $V_p \subseteq V$. 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.

**Algorithm 1** BYZANTINE-AGREEMENT

```
1: while there is no decision, repeat do
2:    For each v ∈ V, cumdev_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-BEN-OR
7:       end for
8:       Run PROCESS-EPOCH
9:    end for
10: end while
```

### 3.1 MODIFIED-BEN-OR

In Ben-Or's 1983 algorithm, a global coin with exponentially small probability is created when each processor flips one coin and their values all coincide. Following [17], we replace these individual coin tosses with a routine GLOBAL-COIN

involving up to $n$ coin tosses per processor. Below is a sketch of the algorithm. The complete MODIFIED-BEN-OR algorithm is found in [17] and in the Appendix B.

We refer to each iteration of the while-loop as an *iteration* of MODIFIED-BEN-OR. 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-BEN-OR, $v_p$ is initialized to be the processor $p$'s input bit for Byzantine agreement.

**Algorithm 2** MODIFIED-BEN-OR (sketch)

1: **while** not decided **do**
2: For a constant number of rounds, exchange initial bits and messages about information received;
3: Let $x$ be the number of messages of support for bit $b$ received:
a) CASE $x > g(n)$: decide $b$;
b) CASE $f(n) < x \leq g(n)$: run GLOBAL-COIN but set $v_p \leftarrow b$;
c) CASE $x \leq f(n)$: $v_p \leftarrow$ GLOBAL-COIN.
4: **end while**

LEMMA 3.1. ([6, 17]) *In an iteration of* MODIFIED-BEN-OR *with $t < n/5$:*

1. *Either there is agreement in the current or next iteration, or all good processors run in* GLOBAL-COIN.
2. *If greater than $4n/5$ good processors start with the same bit value $v$, then every good processors will decide on $v$ in that iteration. In particular, if* GLOBAL-COIN *returns $b$ (from Step 3(b)) to $4n/5$ good processors then every good processor comes to agreement in the next iteration.*

**3.2** GLOBAL-COIN The goal of GLOBAL-COIN (given in the Appendix, Section C) 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 $O(n)$ 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.

The coinflipping occurs in $n$ "rounds". Each processor flips no more than one coinflip per round, and does so after receiving confirmation that its coinflip from the previous round was received by sufficiently many processors. The round number is incremented only if a sufficiently large majority of processors receive the same coinflips for that round from the same large set of processors. The algorithm follows the one in [17] except that the process for incrementing the round number in that paper potentially involves a non-polynomial time computation. We replace that process by the efficiently computable "Spread" protocol in [7]. The full algorithm appears in the Appendix C.

The results of this section are summarized in the following lemma.

LEMMA 3.2. ([17]) *If $t < n/11$ then* GLOBAL-COIN *has the following properties:*

1. *There is a set $S$ of $n - 4t$ good processors which receive $n$ coinflips generated by each of at least $n - 2t$ good processors and receives all but 2 of the coinflips generated by the remaining $t$ good processors, before deciding on the sign of the sum. We use the term "common coins" to refer to this set of at least $n(n-2t)+(n-2)t$ coinflips generated by good processors that are received by all members of $S$.*
2. *All good processors $p$ decide on a sum of the coinflips generated by each processor $q \in V_p$ which is within 3 of the actual sum of coinflips generated by $q$, before deciding on the sum of all the coinflips.*
3. *W.h.p. the absolute value of the sum of coinflips generated by any one good processor is less than $c_3 n^{.5} \ln n - 3$ and if any processor $p$ receives coinflips generated by a processor $q$ with absolute value at least $c_3 n^{.5} \ln n$, $p$ removes $q$ from from $V_p$, for $c_3$ a constant.*

## 4 Detection of Deviation

We describe the information recorded by each processor $p$ after all the iterations of MODIFIED-BEN-OR in an epoch have been completed (Step 7 of BYZANTINE-AGREEMENT) by a matrix. Let $\mathcal{M}_p$ denote the $m \times |V_p|$ matrix such that $\mathcal{M}_p(i, j)$ is the sum of coinflips received in the $i$-th iteration of MODIFIED-BEN-OR from processor $j \in V_p$.

Below we set $\alpha = \sqrt{2n(n-2t)}$, $\beta = \alpha - 2t$. The following lemma is essentially the same as a lemma in [17], except with a small improvement in the constants. The full proof is in the Appendix D

LEMMA 4.1. *Assume that:*

1. *$t < n/36$;*

2. *for each good processor $p$, the number of good processors in $V \setminus V_p$ is no more than $t$; and*

3. *agreement is not achieved in an epoch $e$.*

*Then, w.h.p., there is a set of $.026n$ good processors $P'$ such that for every processor $p \in P'$, there is a set of bad processors $B_{p,e} \subset V_p$ and a set of iterations $I_{p,e}$, $|I_{p,e}| \geq m' = .002m$, such that for every $i \in I_{p,e}$,*

$$\Big| \sum_{j \in B_{p,e}} \mathcal{M}_p(i,j) \Big| > \beta/2.$$

This 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 processors with the requisite sums does not imply that the processors in the set are all bad, but it is a first step used in [17], where that problem is solved in exponential computational time.

In Section 6, we give a polynomial time algorithm for $p$ to measure a processor's contribution to these sums as a means of deciding whether to remove that processor from $V_p$.

## 5 The Complete Neutralizer-Detector Game

We now describe two versions of the Neutralizer-Detector game. Both versions are sufficient in the sense that any strategy for D provides a successful strategy for the Byzantine agreement protocol. The second version is particularly limiting for the algorithm–the adversary may fail in preventing Byzantine agreement in ways other than those that provide a winning strategy for D in the game. The advantage of the second version is that it gives us a way to design and analyze a polynomial time strategy.

Recall the simplified version of the game. The game begins on an $m \times n$ matrix and $m = \Theta(n)$, and it proceeds in epochs with newly generated matrices on a monotone decreasing subset of columns. Let $c$ be a fixed constant. For each epoch, N must pick the values in its columns so that the sum of each row of the matrix is *neutralized*, i.e., the absolute value of each sum of each row of the matrix is less than $cn$. After each epoch $D$ can remove columns so that N will eventually fail. Let $t$ be a small constant fraction of $n$. Initially, N has no columns. Each epoch consists of the following phases.

Phase 1: N may claim over columns, provided that the total number of columns claimed during the game is less than $t$.
Phase 2: All entries in unclaimed columns are independently set to the sum of $n$ independent fair coinflips of value +1 and -1;
Phase 3: N sets the values in its columns.
Phase 4: D may remove columns (for the next epochs), provided that the total number of columns removed during the game is no greater than $2t$.

The game ends when N fails to neutralize a row. The complete versions of the game keep the same framework, but there are some additional specifications described below.

**5.1 General Version** In the most general version, there are $n - t$ players which each independently play the game as D against an adversary N. For each epoch $i$ and all players, there is one $m \times n$ matrix $\mathcal{M}_i$ and one set of columns claimed by N for all $n-t$ instances of the game. However, each player $p$ sees $\mathcal{M}_{i,p}$ which contains a possibly different version of $\mathcal{M}_i$, depending on the columns the player has removed, and the differences described below.

1. All unclaimed entries are generated by the sum of up to $n$ fair coinflips $\in \{+1, -1\}$.

2. Fix an epoch $i$. Let $inS(p, r)$ be a Boolean function determined by N which, for any row $r$, is true for all $p$ in some subset of $n-2t$ players. For any row $r$ of $\mathcal{M}_i$ and for any player $p$, if $inS(p, r)$ is true, the following holds. Let $x$ be the number of columns removed by $p$. Then there are at least $n - 2t - x$ entries in row $r$, in unclaimed columns, such that these entries in $\mathcal{M}_{ip}$ equal the entries in $\mathcal{M}_i$. The remaining up to $t$ values in row $r$ in $\mathcal{M}_{ip}$ are within plus or minus 2 of the values of $\mathcal{M}_i$. If $inS(p, r)$ is false, or a column is claimed by $N$, then the entry in $\mathcal{M}_{i,p}$ for that row and column is within plus or minus 3 of the entry in $\mathcal{M}_i$.

3. A row in $\mathcal{M}_i$ may be *in the right direction*. This is decided independently for each row, by a coinflip with probability 1/2. The results of these coinflips are known to N ahead of time, but not to any player.

4. N fails if $2n/3+1$ good players agree on the sign of the sum of a row that is in the right direction.

**5.2 Specific Version** In the specific version, we use properties of the distribution to arrive at a restriction of the game which makes the design and analysis of a strategy simpler.

1. Fix an epoch $i$. N selects a set of $.026n$ "special" processors in that epoch. For every special processor, $p$, there is a subset $R_p$ of at least $.002n$ rows, with properties described below.

2. If a processor $p$ is special then: 1) $inS(p,r)$ is true (see above) for all $r \in R_p$; 2) the sum of the entries of unclaimed columns in $\mathcal{M}_{i,p}$ have absolute value $> \alpha - \beta/2 - 2t = \beta/2$; and 3) N must neutralize to 0 (or reverse the sign of the sum) for all row $r \in R_p$ for $p$, or else N fails.

3. Processor $p$'s strategy in epoch $i$ can depend only on $\mathcal{M}_{i',p}$, for all $i' \leq i$. In particular, $p$ does not know if it's special for an epoch, or the values of the function $inS$, or which rows are in $R_p$.

4. Each positive (resp. negative) entry $a$ in any matrix, is replaced by $\min\{n^{.5}\log n, a\}$ (resp., $\max\{-n^{.5}\log n, a\}$).

There will be many epochs, possibly all, in which $p$ is not special. Additionally, $p$'s strategy must depend only on the values in the $\mathcal{M}_{i,p}$. We show that each player has a strategy with computation cost of $O(n^3)$ time per epoch, which ends the game in an expected $O(n)$ epochs. The communication cost of the Byzantine agreement protocol is $O(n^2)$ per epoch or $O(n^3)$ overall.

## 6 Spectral Analysis of Coinflips

**Algorithm 3** PROCESS-EPOCH

1: **if** $|\mathcal{M}_p| \geq (\beta/2)\sqrt{m'/t}$ then do **then**
2: Let $\boldsymbol{r_p}$ be the right singular vector of $\mathcal{M}_p$
3: For each $1 \leq i \leq n$, increase **cumdev**$(i)$ by $(\boldsymbol{r_p}[i])^2$
4: For each $1 \leq i \leq n$ remove processor $i$ from $V_p$
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 $\beta/2 = \sqrt{2n(n-2t)}/2 - t$. From Lemma 4.1, let $m$ be the number of iterations in an epoch, let $P'$ be the set of processors $p$ such that $p$ receives all the common coins and for each $p$ there exists a set of $m'$ "good" iterations in the epoch such that the sum of coinflips $r - received$ from bad processors in $V_p$ exceeds $\beta/2$.

The following is a restatement of Theorem 3 from Achlioptas and McSherry [1].

THEOREM 6.1. *[1] Let $\mathcal{M}$ be a random $m$ by $n$ matrix such that $\mathcal{M}(i,j) = r_{ij}$ 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*

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

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

Let $\mathcal{G}$ be a $m$ by $n-t$ matrix based on all coinflips broadcast by the good processors. Specifically, $\mathcal{G}(i,j)$ is the sum of the coinflips broadcast in the $i$-th iteration by the $j$-th good processor, if the absolute value of this sum deviation does not exceed $cn^{.5}\ln n$; else it is 0.

COROLLARY 6.1. *For every $\epsilon > 0$, for $n$ sufficiently large,*

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

*Proof.* Note that $\mathcal{G}(i,j) = r_{ij}$ where $r_{ij}$ are independent random variables with $r_{ij} \in [-cn^{.5}\ln n, cn^{.5}\ln n]$, $E(r_{ij}) = 0$ and $Var(r_{ij}) = \sigma^2 \leq n$. Let $\alpha = 1$ and $\epsilon > 0$ in Theorem 6.1. Then for any positive constant $\epsilon$, 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 a $m$ by $|G \cap V_p|$ matrix, where $\mathcal{G}_p(i,j)$ is the sum of the coinflips r-received by $p$ in the $i$-th iteration from the $j$-th good processor in $V_p$.

LEMMA 6.1. *Assume $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n-t)})$. Then for all $p$, $|\mathcal{G}_p| \leq (6+\epsilon)\sqrt{n(m+n-t)}$.*

*Proof.* Fix a processor $p$ and let $\mathcal{G}'_p$ be the $m$ by $|V_p|$ matrix obtained by omitting columns for $j \notin V_p$ from the matrix $\mathcal{G}$. It is easy to see that $|\mathcal{G}'_p| \leq |\mathcal{G}|$. By Lemma C.5, $|sum_p(q) - sum(q)| \leq 3$. Hence $\mathcal{G}_p = \mathcal{G}'_p + \mathcal{A}$ where all entries of $\mathcal{A}$ are integers between -3 and 3. Clearly, $|\mathcal{A}| \leq 3\sqrt{mn}$. We thus have $|\mathcal{G}_p| = |\mathcal{G}'_p + \mathcal{A}| \leq |\mathcal{G}'_p| + |\mathcal{A}| \leq |\mathcal{G}| + 3\sqrt{mn}$ and the result follows by Corollary 6.1.

For a given processor $p$, let $\mathcal{B}_p$ be an $m$ by $|V_p \cap B|$ matrix where $\mathcal{B}_p(i,j)$ is the sum of coinflips r-received by $p$ in the $i$-th iteration from the $j$-th bad processor in $V_p$.

For the remainder of the paper, we assume $t$ is sufficiently small so that $(6+\epsilon)\sqrt{n(m+n-t)} < (1/k)(\beta/2)\sqrt{m'/t}$, e.g., $t < \frac{\beta^2 m'}{4(6+\epsilon)^2 k^2 nm}$, where $k = 5.45$ and $\epsilon$ any constant.

LEMMA 6.2. *Assume* $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n-t)}$. *Let* $k > 0$. *Then for any processor* $p$ *such that* $|\mathcal{B}_p| \geq (\beta/2)\sqrt{m'/t}$, *we have* $|\mathcal{G}_p| \leq (1/k)|\mathcal{B}_p|$.

*Proof.* By Lemma 6.1, $|\mathcal{G}_p| \leq (6+\epsilon)\sqrt{n(m+n-t)}$. Then we have:

$$\begin{aligned} |\mathcal{G}_p| &\leq (6+\epsilon)\sqrt{n(m+n-t)} \\ &\leq (1/k)(\beta/2)\sqrt{m'/t} \\ &\leq (1/k)|\mathcal{B}_p| \end{aligned}$$

LEMMA 6.3. *For any processor* $p \in P'$, *let* $t_p$ *be the number of bad processors in* $V_p$. *Then*

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

*If* $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n-t)}$, *then* $|\mathcal{G}_p| \leq (1/k)|\mathcal{B}_p|$.

*Proof.* Let $\mathbf{x}$ be a length $t_p$ unit vector, where all entries equal $1/\sqrt{t_p}$. Consider the vector $\boldsymbol{y} = \mathcal{B}\boldsymbol{x}$. Note that for at least $m'$ entries of $\boldsymbol{y}$, the square of the value of that entry is at least $(\beta/2)^2/t_p$. Hence $|\boldsymbol{y}| \geq (\beta/2)\sqrt{m'/t_p} \geq (\beta/2)\sqrt{m'/t}$. The second inequality follows from Lemma 6.2.

Recall that $\mathcal{M}_p$ is the $m$ by $|V_p|$ matrix such that $\mathcal{M}_p(i,j)$ is the sum of the coinflips r-received by processor $p$ in the $i$-th iteration from the $j$-th processor in $V_p$. For simplicity of analysis, we assume that the columns of $\mathcal{M}_p$ are arranged so that the columns for the $t_p$ bad processors in $V_p$ are to the left of the columns for the $n_p - t_p$ 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 $\mathcal{M}$.

We thus let $\mathcal{M}_p = [\mathcal{B}_p\mathcal{G}_p]$. 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 6.4. *Assume* $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n-t)})$. *Then for every processor* $p$ *such that* $|\mathcal{M}_p| \geq (\beta/2)\sqrt{m'/t}$, $|\boldsymbol{g_p}|^2 < |\boldsymbol{b_p}|^2/2$. *In particular, this holds for all* $p \in P'$.

*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 $\mathcal{M}_p\boldsymbol{r_p} = [\mathcal{B}_p\mathcal{G}_p](\boldsymbol{b_p} + \boldsymbol{g_p}) = \mathcal{B}_p\boldsymbol{b_p} + \mathcal{G}_p\boldsymbol{g_p}$. 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:

$$\begin{aligned} |\mathcal{B}_p| &\leq |\mathcal{M}_p| \\ &= \boldsymbol{\ell_p}^T(\mathcal{M}_p\boldsymbol{r_p}) \\ &\leq |\boldsymbol{\ell_p}||\mathcal{M}_p\boldsymbol{r_p}| \\ &\leq |\mathcal{B}_p||\boldsymbol{b_p}| + |\mathcal{G}_p||\boldsymbol{g_p}| \\ &\leq |\mathcal{B}_p|(|\boldsymbol{b_p}| + (1/k)|\boldsymbol{g_p}|) \text{ by Lemmas 6.2 and 6.3} \\ &\leq |\mathcal{B}_p|(\sqrt{2/3} + 1/k) \\ &< |\mathcal{B}_p| \end{aligned}$$

which is clearly a contradiction. In the above inequalities, the third line follows by the Cauchy-Schwartz inequality, and the last line follows for $k > \sqrt{3}/(\sqrt{3} - \sqrt{2}) = 5.45$.

LEMMA 6.5. *With probability at least 1/2, Algorithm 1 will terminate successfully in* $84t$ *epochs, each consisting of* $m = 2n$ *iterations, with resilience* $t < \frac{\beta^2 m'}{4k^2(6+\epsilon)^2 nm} = 2n * .002m/(4 * 30 * 36nm) = .000028n$. *When the algorithm terminates, all processors will achieve Byzantine agreement. The algorithm is Las Vegas with expected* $2 * 84tm < n^2$ *iterations of modified Ben-Or's algorithm for a total of* $O(n^3)$ *rounds of communication and polynomial time computation.*

*Proof.* Note that with probability $1 - 1/(m+n-t) - 1/n^c$, every iteration in an epoch will be such that $|\mathcal{G}| \leq (3+\epsilon)\sqrt{n(m+n-t)}$.

We first claim that no more than $t$ good processors are ever removed from $V_p$ for any processor $p$. First, observe that each processor $v$ is removed from $V_p$ when $\mathbf{cumdev}_p(v) \geq 1$. Each epoch can add no more than 1 to this total for any processor. Hence, the maximum $\mathbf{cumdev}_p$ accrued by a processor before its removal is less than 2. Assume to the contrary that more than $t$ good processors have been removed from $V_p$, then $\sum_{i \in G} \mathbf{cumdev}_p(i) > t$. But in any epoch where processor $p$ adds to the **cumdev** values, it must be the case that $|\boldsymbol{b_p}|^2 \geq 2|\boldsymbol{g_p}|^2$, and so the increase in $\mathbf{cumdev}_p$ values for processors in $B$ is twice the increase of $\mathbf{cumdev}_p$ values for processors in $G$. Thus, $\sum_{i \in B} \mathbf{cumdev}_p(i) > 2t$. Since there are no more than $t$ bad processors, this implies that for at least one bad processor $i$, $\mathbf{cumdev}_p(v) > 2$, giving a contradiction.

We next show that all bad processors will be removed from each $V_p$ for all $p \in G$ in $O(n)$ epochs. By Lemma 6.3, w.h.p. in every epoch which does not terminate, there are $.052 * .048n > .024n = \Theta(n)$ processors

in $P'$ such that $|M_p| \geq |B_p| \geq \beta/2\sqrt{m'/t}$, thus the value of $\sum_i \mathbf{cumdev}_p(i)$ must increase by 1 for each of these processors in $P'$. Thus, $\sum_p \sum_i \mathbf{cumdev}_p(i)$ must increase by $.024n$ in each of these epochs. As shown above, at most $t$ good processors can be removed from $V_p$. Thus, once $\sum_p \sum_v \mathbf{cumdev}_p(v) > n2t$ all bad processors are removed from each $V_p$ for all $p \in G$. If the conditions for Lemma 6.3 hold in each epoch that does not terminate, then there will be $2tn/.024n \leq 84t$ epochs.

Set $c_1 n = 84t$ as the number of epochs needed to remove all bad processors and let $m = 2n > 2c_1 n$. Then with probability at least $1 - c_1 n/(m+n-t) - c'n/n^c > 1/2$, the entire algorithm will successfully run for $c_1 n$ epochs until all bad processors are removed. Then it will succeed w.h.p. in the next epoch. If this fails to occur, it will repeat until Byzantine agreement is decided, making the algorithm Las Vegas. As each epoch contains $O(n)$ iterations of MODIFIED-BEN-OR; each execution of MODIFIED-BEN-OR contains one GLOBAL-COIN which in turn contains $O(n+1)$ rounds of communication, the total communication cost is $O(n^3)$ expected time.

We can analyze computation time per processor as follows. Each round of GLOBAL-COIN requires $O(n^2)$ computation to process up to $n$ coinflips and received coinflip messages. After each epoch the computation of the singular value decomposition requires $O(n^3)$. Thus the total computation time is dominated by the cost of MODIFIED-BEN-OR, for a total expected cost of $O(n^5)$.

This concludes the proof of Theorem 1.

## 7 Conclusion and Future Work

We have described an algorithm to solve Byzantine agreement in polynomial expected communication time and computation time. Our algorithm works in the asynchronous message-passing model, when an adaptive and full-information adversary controls a constant fraction of the processors. 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 $O(n^{2.5})$ as achieved in [17] but with polynomial time computation? An intriguing open question is whether the expected communication time can be brought down to the known lower bound of $\tilde{\Omega}(n)$ or whether Byzantine agreement is intrinsically harder than consensus, in terms of time or step complexity.

## A Appendix

The material here is substantially contained in [17], with the exception of some improved constants, and the use of the Spread routine from [7] in GLOBAL-COIN, making it computable in $O(n^2)$ time.

## B MODIFIED-BEN-OR algorithm

We now describe MODIFIED-BEN-OR, a slight modification of Ben-Or's algorithm for Byzantine agreement [6].

We refer to each iteration of the while-loop as an *iteration* of MODIFIED-BEN-OR. The only change to Ben-Or's protocol is that instead of flipping a private coin, a processor uses a coinflip generated by the algorithm GLOBAL-COIN. The GLOBAL-COIN algorithm takes as an argument the iteration number of MODIFIED-BEN-OR and attempts to generate a fair global coin for that iteration; we describe GLOBAL-COIN later as Algorithm 5.

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-BEN-OR, $v_p$ is initialized to be the processor $p$'s input bit for Byzantine agreement.

The following lemma follows from the result in [6].

LEMMA B.1. (BEN-OR [6]) *In an iteration of* MODIFIED-BEN-OR *with $t < n/5$:*

1. *If greater than $4n/5$ good processors have the same vote value $v$, then every good processors will decide on $v$ in that iteration.*
2. *If a good processor sends $(2, r, v, D)$, then no other good processor sends $(2, r, v', D)$ for $v' \neq v$.*
3. *If at least $2t+1$ D-messages are sent by good processors, then the outcome from* GLOBAL-COIN *is not used and there is a decision in the next iteration.*
4. *If no more than $2t$ D-messages are sent by good processors then all good processors participate in* GLOBAL-COIN.
5. *If* GLOBAL-COIN$(k)$ *returns $v$ to $4n/5$ good processors and no good processor has received at least $t+1$ messages $(2, r, v', D)$ for $v' \neq v$, then every good processor comes to agreement in the next iteration.*

**Algorithm 4** MODIFIED-BEN-OR

```
1:  k ← 1
2:  while not decided do
3:    send the message (1, k, v_p) to all processors;
4:    wait until messages of type (1, k, ∗) are received
      from n − t processors;
5:    if there are more than (n + t)/2 messages of the
      form (1, k, v) then
6:      send the message (2, k, v, D) to all processors;
7:    else
8:      send the message (2, k, ?) to all processors;
9:    end if
10:   wait until messages of type (2, k, ∗) are received
      from n − t processors;
11:   if there are more than (n + t)/2 D-messages of
      the form (2, k, v, D) then
12:     decide v;
13:   else if there are at least t + 1 D-messages
      (2, k, v, D) then
14:     run GLOBAL-COIN(k) but set v_p ← v;
15:   else
16:     v_p ← GLOBAL-COIN(k);
17:   end if
18:   k ← k + 1;
19: end while
```

*Proof.* The proof follows from the correctness of Ben-Or's algorithm and the observation that if no more than $2t$ D-messages are sent by good processors, then no more than $3t \leq (n+t)/2$ D-messages are received by all processors and lines 13-17 apply. Otherwise, if at least $2t+1$ D-messages are sent by good processors, then each processor receives at least $t+1$ D-messages and so only lines 11-14 apply and the output of GLOBAL-COIN is not used.

# C GLOBAL-COIN

The algorithm makes use of the *reliable broadcast* primitive from Bracha [8]. In this primitive, a single player calls *broadcast* for a particularly message $m$, and subsequently, all players may *decide* on exactly one message. The reliable broadcast primitive guarantees the following:

1. If a good player broadcasts a message $m$, then all good players eventually decide $m$.
2. If a bad player $p$ broadcasts a message then either all good players decide on the same message or no good players decide on a message from $p$.

The algorithm assumes that all broadcasts are reliable broadcasts; we use the word *broadcast* to refer to reliable broadcast, and the word *r-received* to refer to deciding on a message which was reliably broadcast. In addition, we define *set-broadcast* to have the properties of reliable broadcast and also have the following additional property.

- A set-broadcast is not r-received by a processor $p$ unless $p$ has received messages from $n-t$ processors participating in the set-broadcast.

The algorithm has the following types of messages.

- *coinflip message* $(p, c, i)$: broadcast by processor $p$ when $p$ generates its $i$-th coinflip that has value $c$
- *received-coinflip message* $(p, q, c, i)$: broadcast by processor $p$ when $p$ r-receives the coinflip message $(q, c, i)$
- *release message* $(p, i)$: sent by processor $p$ only to processor $q$ after $p$ r-receives $n - t$ received-coinflip messages of the form[2] $(*, q, c, i)$
- *received-sum message* $(p)$: broadcast by processor $p$ once it completes the last round of the algorithm. This message consists of $n$ values: for each processor $q$, there is a value giving the sum of all coinflips that $p$ received for $q$

In the algorithm, $i_p$ is the number of coinflips $p$ has generated to completion, and $j_p$ is the number of rounds which $p$ has observed to completion.

## C.1 Analysis of GLOBAL-COIN

LEMMA C.1. *In* GLOBAL-COIN, *every processor will eventually decide a value of the global coinflip.*

*Proof.* We prove this by induction on the number of rounds. We will show that for all $0 \leq j \leq n$, if all good processors reach round $j$, then all good processors will reach round $j+1$. The lemma then follows since a processor decides a value of the global coinflip as soon as it reaches round $n+1$.

For any processor $p$, there are two conditions that must be satisfied for $p$ to advance from round $j$ to round $j+1$. The first is that the processor is not waiting on Step 1(d). The second condition is that the processor is not waiting on Step 2(c).

The first condition will always eventually occur for any processor $p$. To see this, note that if there is some coinflip $c$, and some $k \leq j$, and $p$ has r-received at least $t+1$ received-coinflip messages of the type $(*, b, c, k)$, then at least one good processor has r-received the

[2]The $*$ notation means that an argument can be any value

coinflip message $(b, c, k)$. Thus eventually, $p$ will r-receive the coinflip message $(b, c, k)$. Hence, for the remainder of this proof, we focus solely on the condition of Step 2.

Assume all good processors reach round $x$. We note that if one good processor then reaches round $x + 1$, that all good processors will eventually reach round $x + 1$. To see this, let $p$ be one of the good processors that eventually reach round $x + 1$. This implies that $p$ satisfied the conditions of Step 2, namely, $p$ r-received sets which were set-broadcast by $t + 1$processors and r-received all the messages in the sets. Therefore every message in these set will eventually be r-received by every processor $g$ and $g$ will participate in the set-broadcast of that set. Eventually every good processor will r-receive the same sets and move to round $x + 1$.

We now show that at least one good processor will eventually reach round $x + 1 \leq n + 1$, given that all good processors have reached round $x$. Assume not. Then all good processors are stuck in round $x$ indefinitely. While this is true, for any good processor $p$ that has broadcast coin flip $i \leq x$, the coinflip message $(p, c, i)$ will eventually be r-received by every good processor $q$. Then at least $n - t$ processors $q$ will broadcast the received-coinflip message $(*, p, c, i)$, which will eventually be received by all good processors $q'$, which will send a release message $(q', i)$ to $p$. Thus, $p$ will eventually complete its $i$-th coin toss, for all $i \leq x$.

Assume to the contrary that no processor has completed round $x < n + 1$ and advanced to $x + 1$, and use what we have shown, that all processors will eventually reach round $x$. Then the following will occur: all good processors will broadcast their $x$-th coinflip; the coinflip message $(p, c, x)$ will be r-received by all good processors; all good processors $q$ will broadcast the received-coinflip messages $(*, p, c, x)$; and all processors will r-receive these coinflip messages $(p, c, x)$, and received-coinflip messages, $(*, p, c, x)$.

LEMMA C.2. *There is a set $S$ of processors of size $n-2t$ such that $n-2t$ good processors that r-receive $n$ coinflips from all processors in the set before they set their value of the global coin.*

*Proof.* By Lemma C.1, all processors eventually decide the value of the global coin. Let $p$ be the first good processor to do so. By the condition of Step 2, $p$ has r-received $t+1$ sets, at least one set-broadcast by a good processor. Since the good processors wait to participate in a set-broadcast until they have r-received all the messages in the set, and since set-broadcast requires the participation of at least $n - 2t$ good processors, then $n - 2t$ good processors r-received the $n^{th}$ round coinflips by the same $n - t$ processors ($n - 2t$ of which are good) whose coinflips are reported in that set before $p$ completed the round and set the global coin. A processor which r-receives the $n^{th}$ round coin flipped by a processor must also have r-received its previous coins.

LEMMA C.3. *Consider the coinflip messages broadcast by processors in the set $V \setminus S$, where $S$ is as defined in Lemma C.2. There is a set of $n - 2t$ good processors that r-receive, before they set their value of the global coin, all but possibly two coinflip messages broadcast by each good processor in $V \setminus S$.*

*Proof.* Order the coinflip messages of good processors by when their broadcasts are begun.

Let $b_1$ and $b_2$ be the last two coin flip messages broadcast by processor $B$, where processor $B$ is chosen over all good processor to maximize the time $t$ that $b_1$ was broadcast.

Let $t$ be the time of $b_1$'s broadcast. Consider any other good processor $A$ which broadcasts at least three coinflip messages. All but one of these were broadcast at time no later than $t$. Let $a_1$ and $a_2$ be the last two coinflip messages broadcast by $A$ at time no later than $t$. Let $S_{a_1}$ and $S_{b_1}$ be the sets of processors which broadcast release messages for $a_1$ (resp. $b_1$) before $a_1$ (resp. $b_1$) were completed. Let $R_{b_1}$ be the set of processors which broadcast received-coinflip messages for $b_1$.

Then since the broadcast of $a_2$ occurred by time at most $t$, every processor in $S_{a_1}$ received receive-coinflip messages for $a_1$ from $n - t$ processors by time $t$. Clearly all broadcasts of received-coinflip messages for $b_1$ occurred after time $t$. Since $|S_{a_1}| \geq n - t$ and $|R_{b_1}| \geq n-t$, then $|S_{a_1} \cap R_{b_1}| \geq n-2t$, of which at least $n - 3t$ are good processors.

Note that each processor in $S_{b_1}$ received received-coinflip messages for $b_1$ from $n-t$ processors in $R_{b_1}$, and that there are at least $n-3t$ good processors in $S_{a_1} \cap R_{b_1}$. Thus, at least $n-4t > t$ of the received-coinflip messages for $b_1$ that are received by each processor in $S_{b_1}$ contain the received-coinflip messages for $a_1$ and all previous coinflips by processor A since by Step 1, every broadcast by a processor of a received-coinflip contains a set of all coinflips which have been r-received by that processor, and if this set includes $a_1$, it include all previous coinflips by $A$.

Therefore every processor in $S_{b_1}$ will wait to r-receive a coinflip message for $a_1$ before computing its sums. Hence all processors in $S_{b_1}$ will r-receive all but possibly two coinflip messages of every good processor. This will occur before each of them sets their global coinflip, as it occurs before they send a release message for $b_1$.

Fix a set $S$ of $n-2t$ good processors from Lemma C.2, and another set $S_{b_1}$ of $n-2t$ good processors from Lemma C.3. There are at least $n-4t$ good processors in the intersection of these two sets. This new set of good processors has r-received all coinflips of good processors which were r-received by any processor, except possibly the last two generated by each of $2t$ good processors. We call the coinflips in this set *common coins*.

LEMMA C.4. *There are at least $n(n-2t)$ common coins, and no more than $2t$ coins from good processors, no more than 2 per processor, which are not common. The common coins are known to $n-4t$ good processors.*

LEMMA C.5. *Let $t < n/11$. Then the following hold.*

1. *W.h.p. no good processor will be removed from $V_p$ for any $p$ from Step 11.*
2. *For any good processor $q$, let $sum(q)$ be the sum of all the coin flips broadcast by $q$ during the course of* GLOBAL-COIN. *Then for any good processor $p$, it must be the case that $|sum_p(q) - sum(q)| \leq 3$.*
3. *For any bad processor $q$, let $p_1$ and $p_2$ be good processors that have not eliminated $q$ from $V_{p_1}$ or $V_{p_2}$ in Step 3 of* GLOBAL-COIN, *then it must be the case that $|sum_{p_1}(q) - sum_{p_2}(q)| \leq 2$.*

*Proof.* We begin with part (2). In step 3 of GLOBAL-COIN, $n-t$ received-sum messages are r-received, and at least $n-2t$ such messages must come from good processors. By Lemma C.5, w.h.p., there are no more than $4t$ good processors which are not in $S$ as defined in the statement of that lemma. Thus, in step 3 of GLOBAL-COIN, each processor r-receives $n-t$ received-sum messages, at least $n-5t$ of which are from good processors that know the common coins.

Now fix a good processor $q$ and let $c_{\ell-1}$ and $c_\ell$ be the last two coinflips of processor $q$. By Lemma C.4, there are no more than two coins per processor that are not common and the common coins are known by all but $4t$ good processors. Thus, by the above paragraph, $vote_p(q, sum(q)) + vote_p(q, sum(q) - c_\ell) + vote_p(q, sum(q) - c_\ell - c_{\ell-1}) \geq n - 5t$. Now assume that at the end of Algorithm 5, processor $p$ sets $sum_p(q)$ to be some value $x$ such that $|sum(q) - x| \geq 3$. Then by step 3, $vote_p(q, x-1) + vote_p(q, x) + vote_p(q, x+1) \geq n - 5t$. But since $x-1$, $x$ and $x+1$ are disjoint from $sum(q)$, $sum(q) - c_\ell$, $sum(q) - c_\ell - c_{\ell-1}$, this implies there are at least $2n-5t$ votes distributed across these 6 values. This is a contradiction since $2n - 5t > n$ provided $t < n/10$.

We now show part (1) of the lemma. Let $X$ be the sum of at most $n$ coinflips. The Chernoff bound given in Fact 1 in the following section shows that $Pr(|X| \geq -3 + c_3 n^{.5} \ln n \leq 2e^{(\frac{(3-c_3 n^{.5} \ln n)^2}{2n})} = n^{-c}$ for any $c$ where $c_3$ is a constant dependent on $c$. Thus, by part (2) of the lemma, it must be the case that $|sum_p(q)| \leq c_3 n^{.5} \ln n$.

We now prove part (3). Assume $p_1$ and $p_2$ are good processors that have not removed $q$ from $V_{p_1}$ or $V_{p_2}$ in Step 3 of the algorithm. Let $x_1 = sum_{p_1}(q)$ and $x_2 = sum_{p_2}(q)$ be the values set in Step 3 by $p_1$ and $p_2$ respectively. It must be the case that both $vote_{p_1}(q, x_1-1) + vote_{p_1}(q, x_1) + vote_{p_1}(q, x_1+1) \geq n-5t$ and $vote_{p_2}(q, x_2-1) + vote_{p_2}(q, x_2) + vote_{p_2}(q, x_2+1) \geq n-5t$.

Assume by way of contradiction that $|x_1 - x_2| \geq 3$. Then the integer values $x_1-1$, $x_1$ $x_1+1$, $x_2-1$, $x_2$ $x_2+1$ are all disjoint. We know that the $n-t$ good processors each send the same received-sum message for $q$ to both $p_1$ and $p_2$. Hence, $vote_{p_1}(q, x_1 - 1) + vote_{p_1}(q, x_1) + vote_{p_1}(q, x_1 + 1) + vote_{p_2}(q, x_2 - 1) + vote_{p_2}(q, x_2) + vote_{p_2}(q, x_2 + 1) \leq n + t$. Thus, we have the following inequality $2n - 10t \leq n + t$. This is a contradiction provided that $t < n/11$.

Lemma 3.2 follows immediately from the lemmas above.

## D Analysis of Deviation

The *deviation* of a stream of coinflips generated by a set of processors is the absolute value of the sum of #1's and #-1's in the stream. We refer to the sign of the deviation as its *direction*. Below we set $\alpha = \sqrt{2n(n-2t)}$ and $\beta = \alpha - 2t$.

We first analyze the deviations of the coinflips generated by the processors.

### D.1 Useful lemmas about the distribution of coinflips

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:** 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 $Pr(X > a\sqrt{X})$ converges to $1 - \Phi(a) > (1/a - 1/a^3)(1/\sqrt{2\pi})e^{-a^2/2}$ [Feller in AC]. E.g., $Pr(X > \sqrt{2}\sqrt{N}) > 1/20$.

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

LEMMA D.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 D.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* $e^{(-.638n)^2/(2n^2(1/36))} < e^{-11}$.

**D.2 No agreement implies unusual deviation by bad processors** In this subsection, we assume no more than $t$ good processors have been removed from $V_p$ for any $p$ and show that w.h.p., a failure to come to agreement over a large number of iterations implies there is a large subset of iterations where there are coinflips broadcast by bad processors with unusually high deviation.

For each iteration of MODIFIED-BEN-OR, there is a particular value for the global coin toss (1 or -1) which will result in agreement. We call this the *correct direction*. We now show that for a large majority of processors $p$, there are many iterations with high deviation of coinflips by good processors in $V_p$ in the correct direction.

LEMMA D.3. *Assume that the number of good processors in* $V \setminus V_p$ *is no greater than* $t$ *for all processors* $p$. *Then, with probability at least* $1 - e^{-\Omega(n)}$, *in* $m \geq n$ *iterations of* MODIFIED-BEN-OR, *there are at least* $m' = .048m$ *iterations* $I$ *with the following property. For each iteration* $i \in I$*:*
*(i) the deviation of coinflips of all good processors in iteration* $i$ *is at least* $\alpha$ *in the correct direction; and*
*(ii) there is a set of good processors* $S'$ *of size greater than* $.99n - t$ *such that for all* $p \in S'$, *the set of good processors in* $V \setminus V_p$ *generate coinflips with deviation less than* $\beta/2$ *in the correct direction.*

*Proof.* Fix a processor $p$. Since $V \setminus V_p$ has less than $t$ good processors, Lemma D.2 shows the probability that the deviation of the coinflips of these good processors in $V \setminus V_p$ exceeds $\beta/2$ is less than $e^{-11}$ in any fixed iteration. Hence, in any fixed iteration, the expected number of processors $p$ such that the good processors in $V \setminus V_p$ have deviation exceeding $\beta/2$ is less than $(n-t)e^{-11}$.

Consider the event that at least $ne^{-5} < .01n$ processors $p$ have good processors in $V \setminus V_p$ with deviation exceeding $\beta/2$ in one iteration. By Markov's Inequality, the probability of this event is less than $e^{-6}$. Hence the expected number of iterations in which this event occurs is at most $me^{-6}$. Let $X$ be the number of iterations in which the event occurs. Since each iteration is independent, we can use Chernoff bounds to bound $X$: $Pr(X \geq (1+e^{-2})me^{-6}) = e^{-me^{-10}/3}$. This implies $Pr(X \geq .003m) \leq e^{-\Omega(n)}$.

Let $Y$ be the number of iterations in which all good processors have deviation in the correct direction of at least $\alpha$. From Lemma D.1, $E(Y)$ is at least $m/20$. Using Chernoff bounds, $Pr(Y < (1-e^{-4})m/20) = e^{-me^{-8}/40}$. This implies $Pr(Y < .049m) \leq e^{-\Omega(n)}$. Then by a union bound, $Pr(X < .003m)$ and $Y \geq .049m)$ is $1 - e^{-\Omega(n)}$. But if both $X < .003nm$ and $Y \geq .049m$, then there are at least $Y - X > .048m$ iterations satisfying conditions (i) and (ii).

The next lemma shows that if the conditions above hold, and the deviation of the coinflips by bad processors is low, agreement will result.

LEMMA D.4. *Fix an iteration of* MODIFIED-BEN-OR. *Let* $S$ *be the set from Lemma 3.2 of good processors which receive the common coins in the execution of* GLOBAL-COIN *in that iteration. Let* $G \subseteq S$ *with* $|G| > 4n/5$. *If:*
*(i) the coinflips of all good processors have deviation at least* $\alpha$ *in the correct direction; and*
*(ii) for every* $p \in G$, *the coinflips of good processors in* $V \setminus V_p$ *have deviation less than* $\beta/2$ *in the correct direction; and*
*(iii) for every* $p \in G$ *the coinflips which are r-received by* $p$ *and broadcast by bad processors in* $V_p$ *have deviation less than* $\beta/2$*;*
*then the processors in* $G$ *will agree on a global coin in the correct direction, and all processors will come to agreement in the next iteration of* MODIFIED-BEN-OR.

*Proof.* We assume without loss of generality that the correct direction for the global coin is +1, which corresponds to the bit value 1 in MODIFIED-BEN-OR.

By Statement (1) of Lemma 3.2, the processors in $G$ will receive all coinflips generated by good processors except at most 2 coinflips from each of as many as $t$ good processors. Hence the adversary may cause at most a $2t$ change in deviation in the distribution of these otherwise random coinflips r-received from good processors. If in addition, 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 each processor in $G$ r-receives is less than $\beta/2$, then the sum of the coinflips which each processor in $G$ uses to compute the global coin is greater than $\alpha - \beta - 2t = 0$.

Thus, the global coin will be in the correct direction for all processors in $G$. Hence each processor $p \in G$ will either ignore the global coin and set $v_p = 1$, or will set $v_p$ to the outcome of GLOBAL-COIN which is also 1. Since $|G| > 4n/5$, the next iteration of MODIFIED-BEN-OR will result in Byzantine agreement.

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

**Definitions:** Let $\mathbf{isum}_p(v,i)$ be the sum of coinflips by $v$ r-received by $p$ in iteration $i$. We define the direction in an iteration $i$ for a set $X$ of processors and a processor $p$ as follows: $\mathbf{dir}_p(X,i)$ is 1 if $\sum_{v\in X} \mathbf{isum}_p(v,i) \geq 0$, and $-1$ otherwise.

We define processor $p$'s view of the deviation in an iteration $i$ for a set $X$ of processors as follows:

$$\mathbf{idev}_p(X,i) = |\sum_{v\in X} \mathbf{isum}_p(v,i)| = \sum_{v\in X} \mathbf{isum}_p(v,i)\mathbf{dir}_p(X,i)$$

LEMMA D.5. *Assume that: $t < n/36$; for each good processor $p$, the number of good processors in $V\backslash V_p$ is no more than $t$; and agreement is not achieved in an epoch $e$. Then, w.h.p., there is a set of $.026n$ good processors, $P'$ such that for every processor $p \in P'$, there is a set of bad processors $B_{p,e} \subset V_p$ and a set $I_e$ of greater than $m' = .002m$ of "good" iterations in epoch $e$ such that for every iteration $i \in I_e$, $\mathbf{idev}_p(B_{p,e},i) \geq \beta/2$. Also there is at least one processor which observes this in $.004n$ iterations.*

*Proof.* By Lemma D.3, w.h.p., there is a set $J$ of $.048m$ iterations which satisfy precondition (i) of Lemma D.4 and for each such iteration, there is a set $S'$ of more than $.99n - t$ good processors which satisfy precondition (ii) of Lemma D.4. By Lemma 3.2 part (1), there are no more than $4t$ good processors which are not in $S$ as defined in the statement of that lemma. Thus for each iteration $j \in J$, there is a set $G_j \subseteq S \cap S'$, of more than $.99n - t - 4t = .99n - 5t$ good processors such that precondition (ii) of Lemma D.4 is satisfied for all $p \in G_j$.

By the above argument, if there has been no decision made in $m$ iterations, then precondition (iii) of Lemma D.4 must *not* hold for *any* iteration in $J$. Thus, for every iteration $j \in J$, there must be a set $T_j \subseteq G_j$, $|T_j| \geq (.99n - 5t) - 4n/5 \geq .052n$ for $t < n/36$, such that for every $p \in T_j$, the coinflips broadcast by bad processors in $V_p$ have deviation at least $\beta/2$ in iteration $j$.

We use an averaging argument to show this. For at least $.026n$ good processors $p$, $p$ observes deviation of at least $\beta/2$ for coinflips by a set of bad processors, $B_{p,e}$ in $V_p$ in at least $.026|J|$ iterations. The argument is as follows: There are $.052n|J|$ processor-iteration pairs in which a processor observes $\beta/2$ deviation in the iteration. Hence there is at least one processor which observes $\beta/2$ deviation in $.052|J| > .002m$ iterations. The maximum number of pairs in containing fewer than $.026n$ different processors is less than $.026n|J|$. Assume by contradiction that the remaining less than $n$ processors each appear in fewer than $.026n|J|$ pairs. Then the total number of pairs is less than $.026n|J| + n((.026|J|) = (.052 * .048)n^2$. In the statement of the lemma, setting $P'$ to be this set of good processors completes the proof.

## References


[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 611–618. ACM, 2001.

[2] N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 594–598. Society for Industrial and Applied Mathematics, 1998.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, 2004.

[4] Y. Azar, A. Fiat, A. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *Symposium on Theory of Computing (STOC)*, 2001.

[5] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *The First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[6] M. Ben-Or. Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM New York, NY, USA, 1983.

[7] M. Ben-Or and R. El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[8] G. Bracha. Asynchronous byzantine agreement protocols. *Journal of Information and Computation*, 75(2):130–143, 1987.

[9] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *(PODC)*, 2000.

[10] S. Chakrabarti, B. E. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. Mining the web's link structure. *Computer*, 32(8):60–67, 1999.

[11] B. Chor and C. Dwork. Randomization in Byzantine agreement. *Advances in Computing Research*, 5:443–498, 1989.

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

[13] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proceedings of*

*the 13th international conference on World Wide Web*, pages 403–412. ACM, 2004.

[14] M. Jerrum. Large cliques elude the metropolis process. *Random Structures & Algorithms*, 3(4):347–359, 1992.

[15] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[16] A. Karlin and A. Yao. Probabilistic lower bounds for byzantine agreement and clock synchronization. Unpublished manuscript.

[17] V. King and J. Saia. Byzantine agreement in polynomial expected time. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2013.

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

[19] L. Kučera. Expected complexity of graph partitioning problems. *Discrete Applied Mathematics*, 57(2):193–212, 1995.

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

[21] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web., 1999.

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

[24] H. Zhang, A. Goel, R. Govindan, K. Mason, and B. Van Roy. Making eigenvector-based reputation systems robust to collusion. In *Algorithms and Models for the Web-Graph*, pages 92–104. Springer, 2004.

**Algorithm 5** GLOBAL-COIN

**Assumptions:** Below, $i, j$ are understood to mean $i_p$ and $j_p$. Initially $i, j \leftarrow 0$. Let $c_3$ be a constant which we set in the proof of Lemma C.5.

1: {*Coinflip:*}
a) Whenever $i \leq j$ and $p$ has not yet initiated the $i^{th}$ coinflip, then $p$ flips a coin $c$ and broadcasts the coin flip message $(p, c, i)$ {"$p$ initiates the $i^{th}$ coinflip"} and all the coinflips it has previously r-received.
b) $p$ does not participate in a coinflip broadcast $(q, c, i')$ unless it has r-received $(q, c, i")$ for all $i" < i'$.
c) Whenever $p$ r-receives a coinflip message $(q, c, i')$, then $p$ broadcasts the received-coinflip message $(*, q, c, i')$ and a list of *all* the received-coinflip messages it has r-received in any round.
d) For every received-coinflip message $(q, c, i')$ included on $t+1$ lists which $p$ receives, $p$ waits until it r-receives $(q, c, i')$.
e) Whenever $p$ r-receives $n - t$ received-coinflip messages $(*, q, c, i')$, then $p$ sends to $q$ the release message $(p, i')$.

2: {*Round:*}
a) For round $j$, when there are $n - t$ processors $q$ such that $p$ has r-received $n - t$ received-coinflip messages $(*, q, c, j)$ for the $j^{th}$ round coinflip message sent by each $q$, $p$ broadcasts these messages as a set.
b) $p$ participates in the set-broadcast of a set sent by other processors only if $p$ r-received every one of its messages.
c) $p$ waits until it r-receives sets from $t + 1$ processors such that for every message in these sets, $p$ r-received the same message. Then $p$ increments $j$. {"$p$ completes a round"}

3: {*Terminate:*} If $j = n + 1$ then
a) $p$ broadcasts a received-sum message containing for each processor $q$, the sum of the coin flips that $p$ received from $q$ and waits until receiving such messages $n - t$ other processors.
b) $p$ broadcasts a received-sum message containing for each processor $q$, the sum of the coin flips that $p$ received from $q$.
c) $p$ waits to r-receive received-sum messages from $n - t$ other processors.
d) For each processor $q$ and value $x$ between $-n$ and $n$, $p$ sets $vote_p(q, x)$ to the number of processors from the previous step that claim that the sum of coinflips they received from processor $p$ is equal to $x$.
e) For each processor $q$, $p$ determines if there is a value $-c_3 n^{.5} \ln n \leq x \leq c_3 n^{.5} \ln n$ such that $vote_p(q, x - 1) + vote_p(q, x) + vote_p(q, x + 1) \geq n - 5t$. If so, $sum_p(q) \leftarrow x$, for the smallest such $x$. If not, $q$ is removed from the set $V_p$.
f) p sets the value of the global coinflip to the sign of the sum of the values $sum_p(q)$ over all processors $q \in V_p$. Then $p$ stops broadcasting messages, but continues to participate in the reliable broadcast of messages sent by other processors.