Byzantine Agreement in Polynomial Expected Time

[Extended Abstract]

Valerie King[*]
Dept. of Computer Science, University of Victoria
P.O. Box 3055
Victoria, BC, Canada V8W 3P6
val@cs.uvic.ca

Jared Saia[†]
Dept. of Computer Science, University of New Mexico
Albuquerque, NM 87131-1386
saia@cs.unm.edu

ABSTRACT

In the classic asynchronous Byzantine agreement problem, communication is via asynchronous message-passing and the adversary is adaptive with full information. In particular, the adversary can adaptively determine which processors to corrupt and what strategy these processors should use as the algorithm proceeds; the scheduling of the delivery of messages is set by the adversary, so that the delays are unpredictable to the algorithm; and the adversary knows the states of all processors at any time, and is assumed to be computationally unbounded. Such an adversary is also known as "strong".

We present a polynomial expected time algorithm to solve asynchronous Byzantine Agreement with a strong adversary that controls up to a constant fraction of the processors. This is the first improvement in running time for this problem since Ben-Or's exponential expected time solution in 1983. Our algorithm tolerates an adversary that controls up to a 1/500 fraction of the processors.

Categories and Subject Descriptors

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

General Terms

Theory, Algorithms, Reliability, Security

Keywords

Byzantine Agreement, Distributed Computing, Randomized Algorithms, Consensus

[*]This research was partially supported by an NSERC grant
[†]This research was partially supported by NSF CAREER Award 0644058, NSF CCR-0313160, and an AFOSR MURI grant.

1. INTRODUCTION

How can we build a reliable system out of unreliable parts? Byzantine agreement is fundamental to addressing this question. The Byzantine agreement problem is to devise an algorithm so that n agents, each with a private input can agree on a single common output that is equal to some agent'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 model. The adversary is *adaptive*: it can determine which processors to corrupt 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" [6].

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.

Time in this model is defined to be the maximum length of any chain of messages (see [12, 6]). 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.

The only results known to the authors for this classic model are the works of Ben-Or (1983) [8] and Bracha (1984) [7]. 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 time if $t = \Theta(n)$. As recently as 2006, Ben-Or, Pavlov and Vaikuntanathan [9] 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 [8, 7] 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. Our main result is the following.

Theorem 1. *Let n be the number of processors. There is a $t = \Theta(n)$ such that Byzantine Agreement can be solved in expected time $O(n^{2.5})$ and expected polynomial bits of communication, in the asynchronous message passing model with an adaptive, full-information adversary that controls up to t processors.*

1.1 Technical Overview

We start with Ben-Or's 1983 algorithm. In this algorithm, roughly speaking, the processors take the majority of each others' votes. When there is a sufficiently large majority of processors which agree on the same bit, the adversary cannot affect the outcome. When there is not a sufficiently large majority, some or all of the processors flip their coins. If the processors which flip their coins happen to all flip them to the same value and that value happens to agree with the value held by the processors which do not flip, the algorithm terminates successfully in the next round. Ben-Or's algorithm reduces Byzantine agreement to the problem of generating a mutually agreed upon coinflip. In this sense it is similar to Rabin's global coinflip algorithm [20], which however assumes the existence of a global coin.

We thus follow the technique in the consensus literature of reducing the agreement problem to the problem of producing a commonly agreed upon coinflip (see the survey [2]; we were particularly inspired by the result in [5] for producing consensus in a shared memory model). The consensus problem is equivalent to Byzantine agreement, except that processors taken over by the adversary suffer crash faults and thus no longer send out messages.

In the consensus problem, a prevalent technique for generating a common coin is to have the processors generate and send out many coinflips. If the sum of these coinflips deviates sufficiently far from 0, then if each processor takes the majority value of the coinflips, 1) all processors will obtain the same value; and 2) that value will be a fair coinflip. It suffices to generate n^2 individual coinflips to ensure that with constant probability, the sum generated by these coinflips will have absolute value exceeding the $O(n)$ deviation (from 0) that the adversary can introduce through scheduling of messages and crash faults.

Unfortunately, in Byzantine agreement, the adversary can introduce $\Theta(nt) = \Theta(n^2)$ deviation through adversarially determined "coinflips". Thus, we need a new technique for limiting adversarial deviation. Our basic approach is to run Ben-Or's algorithm multiple times and to add processors with suspiciously large deviation to a "suspect list" so that their coinflips are subsequently ignored.

In particular, over enough iterations of Ben-Or's algorithm, if no decision has occurred, there must be a group of t or fewer processors that have produced coinflips with suspiciously high deviation. These coinflips with high deviation must have been received by enough good processors to prevent a decision. We show that a good processor can use information about these coinflips to maintain its suspect list in the following way. First, if Ben-Or's fails over many iterations, eventually all bad processors will be added to the suspect list. Second, no more than t good processors are ever added to the suspect list.

When all bad processors are added to each good processor's suspect list, agreement is reached within an expected constant number of iterations of Ben-Or's algorithm. One challenge is to show that not too many good processors are added to the suspect list of any good processor.

Paper Organization: The rest of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we present a slightly modified version of the Ben-Or algorithm, *MODIFIED-BEN-OR*, which calls upon a coinflip algorithm we call *GLOBAL-COIN*. The *GLOBAL-COIN* procedure, presented in Section 4, attempts to generate a commonly agreed upon coin. In Section 5, we analyze properties of the streams of coinflips generated and broadcast during multiple calls to *GLOBAL-COIN*. In Section 6, we present and analyze our main algorithm. 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

Below, all discussion concerns the asynchronous model of computation, where the maximum delay is unknown to the algorithm, and time is defined as described in Section 1.

The Byzantine agreement problem was introduced over 30 years ago by Lamport, Shostak and Pease [18]. 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 [14].

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 time exponential in the number of processors n, if $t < n/5$. The algorithm consists of multiple rounds in which each good processor tosses a coin. The running 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.

The resilience (number of faulty processors tolerated) was improved to $t < n/3$ in 1984 by Bracha [7]. The running time remained exponential. This resilience is the best possible [16]. To the authors' knowledge, these two works represent the state of the art for randomized algorithms for Byzantine agreement in the asynchronous model with a strong adversary, i.e., fully adaptive, with full information.

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) [13]. Under these assumptions, more recent work shows that optimal resilience can be achieved [11].

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 [15]. 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.

A similar approach was used, however, to give a Byzantine agreement algorithm with low communication cost, given an adaptive adversary and private channels in [17] . Essentially, the randomness is "elected" but spread among many processors and kept secret through secret sharing. However, this technique does not seem applicable in the situation where there are no private channels.

Randomized consensus (and Byzantine agreement) algorithms are discussed extensively in Aspnes's 2003 survey on the topic [2]. 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 [3].

In 1998, Aspnes showed a $\Omega(n^2/\log^2 n)$ bound on the number of coinflips required for consensus [1]. 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 [4]. 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 [19] 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 IDs of processors when processing messages (notably through suspect lists), our algorithm is not a fully symmetric round protocol.

3. *MODIFIED-BEN-OR* ALGORITHM

We now describe *MODIFIED-BEN-OR*, a slight modification of Ben-Or's algorithm for Byzantine agreement [8].

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 2.

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 [8].

LEMMA 1 (BEN-OR [8]). *In an iteration of* MODIFIED-BEN-OR *with* $t < n/5$*:*

[1]as well as Byzantine agreement

Algorithm 1 *MODIFIED-BEN-OR*

```
 1: $k \leftarrow 1$
 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 $v_p \leftarrow v$;
15:   else
16:     $v_p \leftarrow$ GLOBAL-COIN($k$);
17:   end if
18:   $k \leftarrow k + 1$;
19: end while
```

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.*

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. $\square$

4. *GLOBAL-COIN* ALGORITHM

The goal of *GLOBAL-COIN* (Algorithm 2) is to generate a fair coinflip which is agreed upon by a large fraction of good 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 computes a sum of coinflips received from each processor, and then decides on the sign of the total sum of coinflips received. For a processor p, V_p is a subset of V whose coinflips p uses to try to decide on a value of the global coin. Initially V_p consists of all the processors.

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

Lemma 2. *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, 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.*

The algorithm makes use of the *reliable broadcast* primitive from Bracha [10]. 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. 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 type $(*, q, c, i)$
- *received-sum message*: 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

We assume that every broadcast by a processor p includes all received-coinflip messages that p previously broadcast.

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.

Algorithm 2 *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 later (specifically in the proofs of Lemmas 7 and 14).

```
1: (WAIT STEP) Whenever $p$ has r-received $t+1$ received-coinflip messages of the type $(*, q, c, k)$ for some processor $q$ and for any $k \leq j$, then $p$ waits until it has r-received the coinflip message $(q, c, k)$.
   *The following steps are carried out in any order if $p$ is not waiting:*
2: 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"}.
3: Whenever $p$ r-receives a coinflip message $(q, c, i')$, then $p$ broadcasts the received-coinflip message $(p, q, c, i')$.
4: Whenever $p$ r-receives $n-t$ received-coinflip messages $(*, q, c, i')$, then $p$ sends to $q$ the release message $(p, i')$.
5: Let $S(p, j)$ be a maximum sized set of processors (possibly including $p$) such that for all $a \in S(p, j)$, $p$ has r-received coin-flip message $(a, c, j)$, and for all $b \in S(p, j)$ $p$ has also r-received received-coinflip message $(b, a, c, j)$. Whenever $|S(p, j)| \geq n-t$ then $p$ increments $j$. {"$p$ completes a round"}
6: When $p$ r-receives release messages $(*, i)$ from $n-t$ processors, then $p$ increments $i$. {"$p$ completes a coinflip"}
7: if $j = n+1$ then
8:     $p$ broadcasts a received-sum message containing for each processor $q$, the sum of the coin flips that $p$ received from $q$
9:     $p$ waits to r-receive received-sum messages from $n-t$ other processors
10:    For each processor $q$ and value $x$ between $-n$ and $n$, $p$ sets $vote_p(q, x)$ to be the number of processors from the previous step that claim that the sum of coinflips they received from processor $p$ is equal to $x$.
11:    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 $V_p$.
12:    p decides on the value of the global coinflip, based on 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.
13: end if
```

4.1 Analysis of *GLOBAL-COIN*

Lemma 3. *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 in the WAIT state of Step 1; the second is the condition in Step 5.

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 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 5.

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 condition of Step 5, namely there is a set $S(p,x)$ of size at least $n-t$. Any other processor q will eventually r-receive any message r-received by p. Hence, there will eventually be some set $S(q,x)$ of size at least $n-t$.

We finally show that at least one good processor will eventually reach round $x+1$, given that all good processors have reached round x. Assume not. Then all good processors are stuck in round x indefinitely. Thus, 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 processors q. Then at least $n-t$ processors q will broadcast the received-coinflip message (q, 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 $i \leq x$.

But then eventually all good processors p 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 message (q, p, c, x); and all processors will r-receive these coinflip messages (p, c, x), and received-coinflip messages, (q, p, c, x). Thus for any good processor g there will be a set $S(g, x)$ of size $n-t$. Hence, g will advance to round $x+1$, which is a contradiction. □

LEMMA 4. *There is a set of $n-2t$ good processors that r-receive all n coinflip messages from all processors in some fixed set S of size $n-t$, before they set their value of the global coin.*

PROOF. By Lemma 3, 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 5, p has r-received coinflip messages $(q, *, n)$ from all processors $q \in S(p,n)$ and p also received received-coinflip messages $(s, q, *, n)$ from all processors $s, q \in S(p,n)$ before any other good processor set the global coin value. Note that the coinflip messages $(q, *, n)$ from a processor q also contain all previous coinflip messages $(q, *, x)$ for all $x < n$. Hence, all the processors in $S(p,n)$ have r-received the coinflip messages about all n of each other's coin flips, and have done so before they set their value of the global coin. To complete the proof, we note that there are at least $n-2t$ good processors in $S(p,n)$ and we let $S = S(p,n)$. □

LEMMA 5. *Consider the coinflip messages broadcast by processors in the set $V \backslash S$, where S is as defined in Lemma 4. 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 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.

Therefore every processor in S_{b_1} will wait to r-receive a coinflip message for a_1. 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 4, and another set S_{b_1} of $n-2t$ good processors from Lemma 5. 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 6. *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 7. *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 11 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 9 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 7, 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 9 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_ℓ be the last two coinflips of processor q. By Lemma 6, 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 2, processor p sets $sum_p(q)$ to be some value x such that $|sum(q)-x| \geq 3$. Then by step 11, $vote_p(q, x-1) + vote_p(q, x) + vote_p(q, x+1) \geq n-5t$. 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 11 of the algorithm. Let $x_1 = sum_{p_1}(q)$ and $x_2 = sum_{p_2}(q)$ be the values set in Step 11 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 2 follows immediately from the lemmas above.

5. 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 $\alpha = 2\sqrt{n(n-2t)}$ and $\beta = \alpha - 2t$.

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

5.1 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., $Pr(X > 2\sqrt{N}) \geq (3/8)(1/(\sqrt{2\pi})e^{-2} > 1/32$.

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

Lemma 8. *A set of at least $n(n-2t)$ good coinflips has a deviation of $\alpha = 2\sqrt{n(n-2t)}$ in any specified direction with probability at least 1/32.*

Lemma 9. *A set of no more than nt good coinflips has a deviation of more than $\beta/2 = \sqrt{n(n-2t)} - t$ with probability at most $e^{-(\beta/2)^2/2tn}$. If $t < n/20$, then $\beta/2 > .898n$ and this probability is at most $e^{(.898n)^2/(2n^2(1/20))} < e^{-8} < 1/2980$.*

5.2 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 generated by bad processors with unusually high deviation.

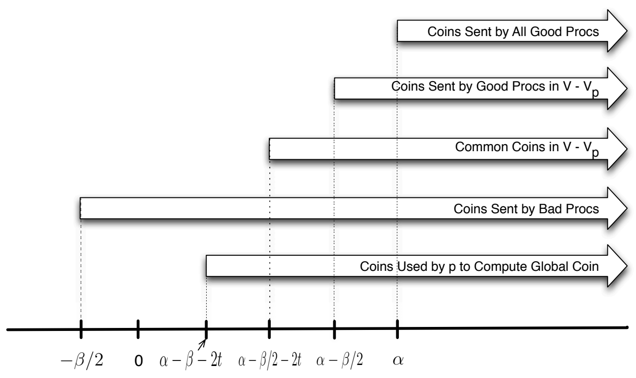
For each iteration of *MODIFIED-BEN-OR*, there is a particular value for the global coin toss (1 or -1) which will result in agreement. We call this the *correct direction*. We now show that for a large majority of processors p, there are many iterations with high deviation of coinflips by good processors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

Lemma 10. *Assume that the number of good processors in $V \setminus V_p$ is no greater than t for all processors p. Then, w.h.p., for sufficiently large constant c, in cn iterations of* MODIFIED-BEN-OR*, there are at least $cn/38$ 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 α in the correct direction; and
(ii) there is a set of good processors S' of size greater than $5n/6$ 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 9 shows the probability that the deviation of the coinflips of these good processors in $V \setminus V_p$ exceeds $\beta/2$ is less than 1/2980 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 $n/2980$.

Consider the event that at least $n/6$ 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 $6/2980 < .003$. Hence the expected number of iterations in which this event occurs is at most $.003cn$. 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: $Pr(X \geq .0035cn) \leq e^{-c'n-1}$, for c' any constant and sufficiently large constant c dependent on c'.

Let Y be the number of iterations in which all good processors have deviation in the correct direction of at least α. From Lemma 8, $E(Y)$ is at least $cn/32$. Using Chernoff bounds, $Pr(Y < cn/33) = e^{-c'n-1}$ for c' any constant and sufficiently large constant c dependent on c'. Then by a union bound, $Pr(X \geq .0035cn$ or $Y \leq cn/33) \leq e^{-c''n}$ for some $c'' > 0$. But if both $X < .0035cn$ and $Y \geq cn/33$, then



Coins Sent by All Good Procs
Coins Sent by Good Procs in V - V$_p$
Common Coins in V - V$_p$
Coins Sent by Bad Procs
Coins Used by p to Compute Global Coin
$-\beta/2$
0
$\alpha-\beta-2t$
$\alpha-\beta/2-2t$
$\alpha-\beta/2$
α

Figure 1: Deviation observed in Lemma 11 by a processor $p \in G$

there are at least $cn/33 - .0035cn > cn/38$ iterations satisfying conditions (i) and (ii). Thus, w.h.p., there are $cn/38$ iterations satisfying condition (i) and (ii). $\Box$

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 11. *Fix an iteration of* MODIFIED-BEN-OR. *Let S be the set from Lemma 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 α 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 generated 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 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. $\Box$

Figure 1 illustrates Lemma 11. This figure shows the deviation observed by some processor p in an execution of *GLOBAL-COIN*. The figure assumes: 1) the correct direction for the global coin is $+1$; 2) p is in the set G defined in Lemma 11; and 3) conditions (i), (ii) and (iii) of the lemma all hold.

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 12. *Assume that: $t < n/12$; 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 an epoch e. Then, w.h.p., there is a set of $n/82$ 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 $cn/(38 * 82)$ iterations in epoch e such that for every iteration $i \in I_e$, $\mathbf{idev}_p(B_{p,e}, i) \geq \beta/2$.*

Proof. By Lemma 10, w.h.p., there is a set J of $cn/38$ iterations which satisfy precondition (i) of Lemma 11 and for each such iteration, there is a set S' of more than $5n/6$ good processors which satisfy precondition (ii) of Lemma 11. By Lemma 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 $5n/6 - 4t$ good processors such that precondition (ii) of Lemma 11 is satisfied for all $p \in G_j$.

By the above argument, if there has been no decision made in *cn* iterations, then precondition (iii) of Lemma 11 must *not* hold for *any* iteration in J. Thus, for every iteration $j \in J$, there must be a set $T_j \subseteq G_j$, $|T_j| \geq (5n/6 - 4t) - 4n/5 \geq n/40$, such that for every $p \in T_j$, the coinflips generated by bad processors in V_p have deviation at least $\beta/2$ in iteration j.

By an averaging argument, for at least $n/82$ 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 $cn/(38 * 82)$ iterations in J. The argument is as follows: There are $cn^2/(38 * 40)$ processor-iteration pairs in which a processor observes $\beta/2$ deviation in the iteration. The maximum number of pairs in which less than $n/82$ processors can appear is less than $(n/82)(cn/38)$. Assume by contradiction that the remaining less than n processors each appear in less than $cn/(38 * 82)$ pairs. Then the total number of pairs is less than $(n/82)(cn/38) + n(cn/(38 * 82)) = 2cn^2/(38 * 82) < cn^2/(38 * 40)$. In the statement of the lemma, setting P' to be this set of good processors completes the proof. $\Box$

The above lemma shows that for every processor p and epoch e, there exists a set of bad processors $B_{p,e} \subseteq V_p$ with high deviation in many iterations. Unfortunately, there may be other sets which share this property, so it is not clear that p can simply find any subset of processors with the requisite deviation and then remove that subset from V_p.

In the next section, we show that p can use any subset it finds to have sufficient deviation as a starting point. But then p does not remove all processors in that subset from

V_p. Instead, we give a method for p to measure a processor's contribution to the deviation of this subset as a means of deciding whether to remove that processor from V_p.

6. ELIMINATING PROCESSORS

To perform the elimination procedure every processor p must keep track of the observed deviation of every processor v in every run of *GLOBAL-COIN*. Recall that *GLOBAL-COIN* is run once per every iteration of *MODIFIED-BEN-OR*, except the final iteration. Further recall that each processor p maintains a view $V_p \subseteq V$ of the current processors whose coinflips it will use to determine the outcome of *GLOBAL-COIN*. The remaining coinflips are discarded. V_p is updated at most once during every cn iterations of *MODIFIED-BEN-OR*. We refer to these cn iterations as an *epoch.*

[**Jared:** *We should write $I_{p,e}$ here since I_e can differ based on p. Kind of hate to make the notation more cumbersome though - maybe we can let these subscripts be implicit or write as $B(p, e)$ or something?*] During an epoch e, when and if p determines there is a set $B_{p,e} \subseteq V_p$ of no more than t processors and a set of $cn/(38*82)$ iterations I_e such that for each iteration $i \in I_e$, $\mathbf{idev}_p(B_{p,e}, i) \geq \beta/2$, p also determines for each processor $v \in B_{p,e}$ the individual *epoch deviation* of the processor v. This is the deviation of the coinflips p has received from v over iterations in I_e where the direction of the deviation in each iteration is determined by the direction of the deviation of $B_{p,e}$ in that iteration. In particular, the epoch deviation $\mathbf{edev}_p(v, I_e, B_{p,e}) = \sum_{i \in I_e} \mathbf{isum}_p(v, i)\mathbf{dir}_p(B_{p,e}, i)$. Once these I_e iterations have occurred, p waits until the start of the next epoch to again attempt to detect deviation for processors.

For every processor v, processor p maintains the cumulative sum of its **edev** over all epochs: $\mathbf{cumdev}_p(v)$. When $\mathbf{cumdev}_p(v) \geq 2(c_3 n^{.5} \ln n)(cn/(38*82))$, v is removed from V_p.

Algorithm 3 Main Algorithm

```
1: while there is no decision, repeat do
2:   For each $v \in V$, $\mathbf{cumdev}_p(v) \leftarrow 0$
3:   $V_p \leftarrow$ the set of all processors
4:   for $e = 1$ to $c_1 n$ {"$p$ runs epoch $e$"} do
5:     $Found \leftarrow FALSE$.
6:     for $i = 1$ to $cn$ do
7:       $p$ runs MODIFIED-BEN-OR
8:       if $Found = FALSE$ and $p$ finds a set $B_{p,e}$ of
         processors of size at most $t$ and a set of $cn/(38*82)$ iterations $I_e$ in epoch $e$ such that for every
         iteration $i \in I_e$, $\mathbf{idev}_p(B_{p,e}, i) \geq \beta/2$ then
9:         for each $v \in B_{p,e}$ do
10:          $\mathbf{cumdev}_p(v) \leftarrow \mathbf{cumdev}_p(v) + \mathbf{edev}_p(v, I_e, B_{p,e})$.
11:          if $\mathbf{cumdev}_p(v) \geq 2(c_3 n^{.5} \ln n)(cn/(38*82))$
             then remove $v$ from $V_p$
12:        end for
13:        $Found \leftarrow TRUE$.
14:      end if
15:    end for
16:  end for
17: end while
```

6.1 Analysis of Elimination Procedure

We show that the elimination procedure will enable good processors to successfully remove bad processors, without removing too many good processors.

Let *Bad* be the set of bad processors and *Good* be the set of good processors. For a good processor p, a set of processors S, and an epoch e, let $\mathbf{cumdev}_p(S, e)$ be the sum over all processors $q \in S$ of the amount added to the variable $\mathbf{cumdev}_p(q)$ in epoch e.

LEMMA 13. *Assume Algorithm 3 runs for at most a polynomial number of epochs and $t < n/500$. Then w.h.p., for any good processor p and any epoch e, $\mathbf{cumdev}_p(Good, e) \leq (\beta/5)cn/(38*82)$.*

PROOF. Let $c_2 = c/(38*82)$. Assume that in some epoch e, p finds a set of processors $B_{p,e}$ and a set of $c_2 n$ iterations I_e such that for every iteration $i \in I_e$, $\mathbf{idev}_p(B_{p,e}, i) \geq \beta/2$. We note that p only increases $\mathbf{cumdev}_p(q)$ for processors $q \in B_{p,e}$. Therefore we would like to bound the amount added for all $q \in Good \cap B_{p,e}$.

Let $G = Good \cap B_{p,e}$ and let X be a random variable that is the sum of all coinflips generated by processors in G during epoch e. By Lemma 5, the sum of coins r-received from a good processor in any iteration differs by at most 3 from the sum of the coinflips which were generated by that processor. Recall that $\beta = 2\sqrt{n(n-2t)} - 2t$. Thus, if $X \leq (\beta/6)c_2 n$, this implies that $\mathbf{cumdev}_p(G, e) \leq X + 3tc_2 n \leq (\beta/6 + 3t)c_2 n \leq (\beta/5)c_2 n$ when $3t < \beta/30$ which occurs when $t < n/46$. Further note that the probability that $X \geq (\beta/6)c_2 n$ is maximized when all processors in G generate all n coinflips for each iteration in I_e. Thus, we will pessimistically assume this is the case throughout the proof.

We now fix a set G, a set I_e and a mapping d from iterations in I_e to directions of the deviation $\{-1, 1\}$. Since X is the sum of independent trials, by Fact 1, and the fact that $\beta^2 \geq 1.9n^2$ (which holds if $t \leq n/10$), we have:

$$\begin{aligned} Pr(X \geq (\beta/6)(c_2 n)) &\leq e^{-(c_2 n \beta/6)^2/2(|G|c_2 n^2)} \\ &\leq e^{-c_2 \beta^2/(72|G|)} \\ &\leq e^{-.026 c_2 n^2/t} \end{aligned}$$

There are $\binom{cn}{cn/(38*82)} \leq (38*82e)^{cn/(38*82)}$ ways to pick the iterations I_e; at most $\sum_{i=1}^{t} \binom{n}{i} \leq 2^n$ ways to pick the set G; and $2^{c_2 n}$ ways to pick the mapping d. Let ξ be the event that $X \geq (\beta/6)(c_2 n)$ for *any* sets G and I, and mapping d. Then by a union bound, we have the following.

$$\begin{aligned} Pr(\xi) &\leq (38*82e)^{c_2 n} 2^n 2^{c_2 n} e^{-.026 c_2 n^2/t} \\ &\leq e^{9c_2 n} e^n e^{c_2 n} e^{-.026 c_2 n^2/t} \\ &\leq e^{11 c_2 n - (.026 c_2 n^2/t)} \\ &\leq e^{-\Omega(n)} \qquad \text{Setting } n/t \geq 500 \end{aligned}$$

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

LEMMA 14. *With high probability, for all processors p, the number of good processors removed from V_p is no more than t.*

PROOF. Suppose a processor p detects a set $B_{p,e}$ as defined in Algorithm 3 in some epoch e, for iterations I_e. From

Lemma 13, w.h.p., $\mathbf{cumdev}_p(Good \cap B_{p,e}, e) < (\beta/5)cn/(38*82)$. Since the total deviation assigned in the epoch is at least $\beta/2$ for each of the $cn/(38*82)$ iterations, this implies that $\mathbf{cumdev}_p(Bad \cap B_{p,e}) > (3\beta/10)(cn/(38*82))$. Hence, the amount of deviation accrued by bad processors in V_p is greater than 3/2 the amount accrued by good processors in V_p in each epoch in which deviation is accrued.

Let G_p be the set of good processors that have been removed from V_p. For a processor p and a set of processors, S, let $\mathbf{cumdev}_p(S) = \sum_{v \in S} \mathbf{cumdev}_p(v)$. If $|G_p| > t$ then $\mathbf{cumdev}_p(G_p) \geq 2(c_3 n^{.5} \ln n)t(cn/(38*82))$. This implies: **equation (1):** $\mathbf{cumdev}_p(Bad) > 3(c_3 n^{.5} \ln n)t(cn/(38*82))$.

Each processor v is removed from V_p when $\mathbf{cumdev}_p(v) \geq 2(c_3 n^{.5} \ln n)(cn/(38*82))$. Each iteration can add no more than $c_3 n^{.5} \ln n$ deviation to a processor (see Lemma 7). Thus, each epoch, including a processor's last before its removal, adds no more than $(c_3 n^{.5} \ln n)(cn/(38*82))$ to $\mathbf{cumdev}_p(v)$. Hence, the maximum $\mathbf{cumdev}_p$ accrued by a processor before its removal is less than $3(c_3 n^{.5} \ln n)(cn/(38*82))$. This together with the fact that there are at most t bad processors contradicts equation (1). □

Lemma 15. *With high probability, Algorithm 3 will terminate in $\tilde{O}(n^{.5})$ epochs ($\tilde{O}(n^{2.5})$ time). When the algorithm terminates, all processors will achieve Byzantine agreement. The algorithm is Las Vegas with expected runtime that is $\tilde{O}(n^{2.5})$.*

Proof. Let X be the sum over all processors p and q of the value $\mathbf{cumdev}_p(q)$. By the argument in the proof of Lemma 14, for any processors p and q, $\mathbf{cumdev}_p(q) \leq 3(c_3 n^{.5} \ln)(cn/(38*82))$. Summing over all n^2 pairs of processors, we see that $X \leq n^2(3c_3 n^{.5} \ln n)(cn/(38*82)) = \tilde{O}(n^{3.5})$. By Lemma 12, w.h.p., in every epoch in which the algorithm does not terminate, the value of X must increase by at least $(n/82)(\beta/2)(cn/(38*82)) = \Theta(n^3)$. Thus, w.h.p., the total number of epochs in which X can increase is $\tilde{O}(n^{.5})$.

By the contrapositive of Lemma 12, w.h.p., if there is an epoch in which X does not increase, Byzantine agreement must be achieved in that epoch. Hence, w.h.p., the algorithm achieves Byzantine agreement in $O(n^{.5})$ epochs.

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

This concludes the proof of Theorem 1.

7. CONCLUSION AND FUTURE WORK

We have described an algorithm to solve Byzantine agreement in polynomial expected 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 which is detectable by individual nodes. Essentially, this reduces the network communication problem to an individual computation problem. This suggests a new paradigm for secure distributed computing: the design of algorithms that force attackers into behavior that is statistically deviant and detectable.

Our result leaves much room for improvement, in terms of the resilience and expected time. First, we did not work hard at reducing the resilience which now stands at $n/500$. Can we increase this value? Second, it is not clear whether the expected 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. Finally, we believe that we can reduce the *computational* cost of our algorithm to polynomial time, and are actively working on this problem.

Acknowledgements: This authors would like to thank the Banff International Research Station (BIRS). This work was initiated at the BIRS Workshop 12w122, "Probabilistic Versus Deterministic Techniques for Shared Memory Computation".

8. REFERENCES

[1] J. Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *Journal of the ACM*, 45(3):415–450, 1998.

[2] J. Aspnes. Randomized protocols for asynchronous consensus. *Journal of Distributed Computing*, 16:165–175, 2003.

[3] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, 1990.

[4] H. Attiya and K. Censor. Lower bounds for randomized consensus under a weak adversary. In *Principles of Distributed Computing (PODC)*, pages 315–324, 2008.

[5] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. *Journal of the ACM*, 55(5), 2008.

[6] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*, page 14. John Wiley Interscience, March 2004.

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

[8] M. Ben-Or. Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols. In *Principles of Distributed Computing (PODC)*, pages 27–30, 1983.

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

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

[11] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *ACM Symposium on Theory of Computing (STOC)*, 1993.

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

[13] P. Feldman and S. Micali. Byzantine agreement in constant expected time (and trusting no one). In *Foundations of Computer Science (FOCS)*, pages 267–276, 1985.

[14] M. Fischer, N. Lynch, and M. Paterson. Impossibility

of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[15] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Scalable algorithms for byzantine agreement and leader election with full information. *ACM Transactions on Algorithms(TALG)*, 2009.

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

[17] V. King and J. Saia. Breaking the O(n^2) bit barrier: Scalable byzantine agreement with an adaptive adversary. In *Principles of Distributed Computing (PODC)*, 2010.

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

[19] A. Lewko. The contest between simplicity and efficiency in asynchronous byzantine agreement. *Distributed Computing*, pages 348–362, 2011.

[20] M. Rabin. Randomized Byzantine generals. In *Foundations of Computer Science (FOCS)*, pages 403–409, 1983.