Scalable Byzantine Agreement

Clifford Scott Lewis * Jared Saia *

Abstract

This paper gives a scalable protocol for solving the Byzantine agreement problem. The protocol is scalable in the sense that for Byzantine agreement over n processors, each processor sends and receives only $O(\log n)$ messages in expectation. To the best of our knowledge this is the first result for the Byzantine agreement problem where each processor sends and receives $o(n)$ messages. The protocol uses randomness and is correct with high probability. [1] It can tolerate any fraction of faulty processors which is strictly less than 1/6. Our result partially answers the following question posed by Kenneth Birman: "How scalable are the traditional solutions to problems such as Consensus or Byzantine Agreement?" [5].

1 Introduction

Peer-to-peer (p2p) networks have emerged for a wide range of applications including data-sharing (e.g. Napster [32], Gnutella [28], Kazaa [30] and Morpheus [31]), computation (e.g. SETI@home [33], FOLDING@home [27] DataSynapse [26], NetBatch [22]), collaboration (e.g. Groove Networks [29]), internet infrastructure systems (e.g. I3 [23]) and distributed storage (e.g. FARSITE [1]). Distributed computation is an integral part of many of these p2p systems [33, 27, 26, 22, 1]. Some of these systems use a trusted third party to initiate or direct the computation. However in many cases, a trusted third party may not be available and so completely distributed algorithms are required.

Unfortunately, distributed computation by a p2p network is vulnerable to attack. Because p2p systems generally lack admission control, there can be large numbers of malicious peers in the system at any time. There are examples where such malicious peers acting alone or in concert have wreaked havoc on a p2p system [6, 35, 7]. For this reason, p2p systems which perform distributed computation need algorithms which work even in the face of malicious attack. In particular, these algorithms must be both scalable *and* attack-resistant.

Towards this end, we consider the problem of designing a *scalable* solution to the Byzantine agreement problem. The Byzantine agreement problem [19, 14] is one of the most well-studied problems in distributed computation and is a building block in many secure distributed protocols. Unfortunately, to the best of our knowledge, all previous solutions to this problem require each processor to send a number of messages which is at least linear in the total number of processors. This is unacceptable for a p2p system, where the number of processors can easily be on the order of hundreds of thousands. In fact, for a p2p system to be scalable, we generally require all resource costs per peer to be polylogarithmic in the number of peers.

This paper gives a protocol which solves the Byzantine Agreement problem while requiring each processor to send only $O(\log n)$ messages in expectation. To the best of our knowledge this is the first result for this problem which requires each processor to send $o(n)$ messages. Our protocol uses randomness and is correct with high probability. It can tolerate any fraction of faulty processors which is strictly less than 1/6.

*Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386; email: {csl, saia}@cs.unm.edu. This research was partially supported by NSF grant CCR-0313160 and Sandia University Research Program grant No. 191445.

[1] In particular, the probability of failure can be made to be $\frac{1}{n^c}$, for any fixed constant c.

A major tradeoff we make to achieve scalability is that our protocol is correct only with high probability. We feel that this tradeoff is justified for two reasons. First, one can not achieve scalability without making this tradeoff. Consider any protocol where each processor receives $o(n)$ messages total. Because a constant fraction of the processors are faulty, there is always some probability that all of the messages received by a fixed processor are from faulty processors. If this event occurs, the faulty processors can force the protocol to fail. Second, the probability of error for our protocol can be made small enough to be essentially imperceptible on a human scale, even for reasonably sized constants. For example, if $n = 100,000$, the fraction of faulty processors is $1/100$, and the constant C in step 1 of the protocol (given in Figure 1) is 600, then the probability of protocol failure is no more than $9 * 10^{-31}$. Note that the number of milliseconds in a human life is less than 10^{13}. Thus, if the protocol runs once per millisecond in this scenario, we would expect at least 10^{17} human lifetimes to pass between failures.

1.1 Related Work

The Byzantine agreement problem was introduced by Pease, Shostak and Lamport [19]. Fischer, Lynch and Paterson show that there is no deterministic protocol to solve the asynchronous Byzantine agreement problem, even if there is only a single processor fauly [11]. Moreover, Fischer and Lynch [12] showed that even in the synchronous communication model, any deterministic protocol requires a linear number of rounds to reach agreement in the worst case.

Rabin [20] and Ben-Or [3] present protocols which overcome this limitation by using randomness. These protocols are always correct and require only a constant number of rounds in expectation. They can also be shown to achieve consensus even in the asynchronous setting. Several subsequent papers improve on these initial results by reducing time complexity, message complexity, bit complexity, and by increasing the fraction of faults that can be tolerated [25, 4, 8, 9]. However, both in Rabin and Ben-Or's and all subsequent protocols, each processor is required to send and receive $O(n)$ messages in expectation.

Our protocol is very similar to Rabin's protocol. The only major difference is that our protocol uses random sampling. While in each round of Rabin's protocol, each processor takes input from all other processors, in our protocol, in each round, each processor takes input from a small random sample of all the other processors. Our main contribution is showing that even with this random sampling, we can still obtain correctness with high probability.

2 The Scalable Byzantine Agreement Protocol

2.1 The Byzantine Agreement Problem

We now formally describe the Byzantine agreement problem. We start with n processors each having an initial binary input. Some fraction of these processors may be faulty. There are no restrictions on the actions of the faulty processors: they may fail to act, act inconsistently or act in collaboration. The correct processors are those which follow the protocol. The correct processors are trying to ensure that at the end of the protocol the following two properties hold:

1. All correct processors have the same output result.

2. If all correct processors have the same input, b_i, then they will all have b_i as an output.

2.2 The Protocol

Our protocol, which we call Scalable Byzantine Agreement, is presented in Figure 1. This protocol is very similar to Rabin's randomized protocol [20], as presented in [17]. In our protocol, each processor first takes input from a small random sample of the processors: $O(\log n)$ processors chosen uniformly at random. Then the value of a global coin flip is established and used to set a threshold for the round. Each processor, based on its sampled inputs, calculates its estimate of the number of correct processors which have the majority vote. The processor sets its vote according to whether or not this estimate exceeds

the threshold chosen for that round. The protocol continues until all processors have crossed the highest threshold.

2.3 Model Assumptions

Our protocol operates under the same model as Rabin's protocol [20]. In this section, we describe the assumptions of this model. We emphasize that these are the same assumptions made in Rabin's model. In Section 3, we show how to adapt our protocol for a peer-to-peer model and thereby remove or relax most of these assumptions.

- *Trusted IDs*: A trusted authority establishes processor identifications.
- *Secure Communication*: There is a secure communication channel between any pair of processors.
- *Bounded Communication Time*: There is an upperbound, U, on the amount of time it takes to send a message from any processor to any other processor. If a processor takes more than $2U$ time to reply to a sent message then that processor is considered faulty. Each processor waits $2U$ time between step 2b and step 2d of the protocol. (We assume that communication time dominates computation time for the processors.)
- *Knowledge of* $\ln n$: Each processor knows the quantity $\ln n$
- *Ability to Choose a Random Processor*: Each processor can choose a processor uniformly at random from the set of all processors. (This gives us the ability to select a set of $C \ln n$ processors uniformly at random)
- *Global Coin Toss*: At each step there is a global coin toss that a trusted third party performs. The coin toss results in heads with probability 1/2 and tails with probability 1/2 and this result is correctly sent to all the processors.

2.4 Properties of the Protocol

Theorem 1 presents the key properties of the Scalable Byzantine Agreement protocol.

Theorem 1. *The Scalable Byzantine Agreement protocol has the following properties:*

- *The protocol can tolerate an* f_T *fraction of faulty nodes for any* $f_T < 1/6$
- *The expected number of rounds is no more than* 3 *and the expected number of messages sent and received by a processor in the protocol is no more than* $6C \ln n$. [2]
- *With probability at least* $1 - 9n^{1-2(\frac{1}{14}-\frac{3}{7}f_T)^2C}$, *the protocol is correct, i.e:*
  - *all correct processors will have the same output result*
  - *if all correct processors have the same input,* b_i, *then they will all have* b_i *as an output.*

The proof of Theorem 1 appears in Section 4. Figure 2 gives some examples of the constants involved in our protocol. The figure shows that even for reasonably small sample sizes, we can achieve a probability of error which is extremely small.

[2] where C is any constant chosen such that $C \ln n$ is an odd number.

Input: A value b_i.
Output: A decision d_i.

The threshold values are $G = (1 - f_t - \alpha)n$, $H = (1 - 2f_t - 4\alpha)n$, and $L = (1 - 3f_t - 7\alpha)n$, where $\alpha = 1/14 - (3/7)f_T$ (which implies that $L = 1/2$). *coin* is the result of a random global coin flip.

1. $vote_i \leftarrow b_i$;
2. For each round, do:
   (a) Select $C \ln n$ processors uniformly at random with replacement from which to receive input (C is a constant described in Theorem 1).
   (b) Request input from the selected processors;
   (c) Send $vote_i$ to the processors which request it;
   (d) $maj_i \leftarrow$ majority value (0 or 1) among votes received.
   $m_i \leftarrow$ the number of occurrences of maj_i among the votes received
   (e) $M_i \leftarrow m_i * \frac{n}{C \ln n}$ (M_i is an estimate of the number of correct processors that have the majority vote);
   (f) if $coin = heads$ then $threshold \leftarrow L$;
   else $threshold \leftarrow H$;
   (g) if $M_i \geq threshold$ then $vote_i \leftarrow maj_i$;
   else $vote_i \leftarrow 0$;
   (h) if $M_i \geq G$ then set d_i to maj_i permanently;

Figure 1: Scalable Byzantine Agreement Protocol

3 Peer-to-Peer Adaptations

In this section, we adapt our protocol to a peer-to-peer model. We assume that there is a collection of n peers which will perform our protocol. We further assume that these peers are connected in a Distributed Hash Table(DHT); in particular, we will assume they are connected in a DHT which is robust to Byzantine faults. Any such DHT will suffice; we will use the "Simple Fault Tolerant DHT" described by Naor and Weider [18]. This DHT ensures secure communication from any peer x to any other peer y, provided that x knows y's ID. The DHT ensures this property with high probability even if each peer in the network independently suffers a Byzantine fault with probability $p < 1/2$. Sending a message from peer x to peer y in the network has $O(\log n)$ latency and requires $O(\log^2 n)$ messages total. Further each peer in their network has degree $O(\log n)$.

In the following itemized list, we address each of the asssumptions made in Section 2.3 in order to adapt our protocol to the peer-to-peer model. We will see that our protocol when run among a set of n peers connected in the "Simple Fault Tolerant DHT" will have the following resource costs: each peer will have a latency of $O(\log n)$ per round and each peer will send $O(\log^3 n)$ messages per round. We note that all of the messages sent will be small in size.

- *Trusted ID's*: All peer-to-peer systems implicitly make the assumption that a trusted authority establishes peer IDs. In fact, Douceur shows that without this assumption, a system is always open to "Sybil Attacks" [10]. Under this attack, a single faulty agent presents multiple peer identities and thereby gains control of an arbitrarily large fraction of the peers in the system. With such control, the single faulty agent can then completely disrupt the system. An identification authority is needed to avoid this type of attack. The authority can be an explicit certification agency like Verisign [13].

n	sample size	failure probability
10^5	$2.3 * 10^3$	$9 * 10^{-4}$
10^6	$2.6 * 10^3$	$9 * 10^{-4.8}$
10^7	$3.2 * 10^3$	$9 * 10^{-5.6}$
10^8	$3.6 * 10^3$	$9 * 10^{-6.4}$

n	sample size	failure probability
10^5	$4.6 * 10^3$	$9 * 10^{-13}$
10^6	$5.2 * 10^3$	$9 * 10^{-15.6}$
10^7	$6.4 * 10^3$	$9 * 10^{-18.2}$
10^8	$7.2 * 10^3$	$9 * 10^{-20.8}$

n	sample size	failure probability
10^5	$6.9 * 10^3$	$9 * 10^{-22}$
10^6	$7.8 * 10^3$	$9 * 10^{-26.4}$
10^7	$9.6 * 10^3$	$9 * 10^{-30.8}$
10^8	$1.1 * 10^4$	$9 * 10^{-35.2}$

n	sample size	failure probability
10^5	$9.2 * 10^3$	$9 * 10^{-31}$
10^6	$1.0 * 10^4$	$9 * 10^{-37.2}$
10^7	$1.3 * 10^4$	$9 * 10^{-43.4}$
10^8	$1.4 * 10^4$	$9 * 10^{-49.6}$

Figure 2: Sample size and protocol failure probability for various n when $f_T = 1/100$. The top left table has $C = 200$; the top right table has $C = 400$; the bottom left table has $C = 600$; and the bottom right table has $C = 800$.

More simply, it can be implicit: peer IDs can be a hash of the peer IP address (e.g. as in Chord [24], CAN [21] and Tapestry [34]), or cryptographic keys embedded in device hardware (e.g. as in the EMBASSY platform [15]). Any type of trusted, easily verifiable peer ID will suffice for our protocol.

- *Secure Communication*: In our protocol, we require that any two procesors can communicate securely. The "Simple Fault Tolerant DHT" fulfills this requirement. When run under this DHT, each peer will have $O(\log n)$ latency and send $O(\log^3 n)$ messages in steps 2b and 2c of our protocol.

- *Bounded Communication Time*: We note that we can decrease latency at the expense of fault-tolerance in the following way. For some constant U, let f_U be the fraction of peers for which communication latency can be greater than U. We can "fold" these peers into the set of faulty peers. In other words, if we increase f_t by f_U, then each processor need wait only $2U$ time between step 2b and step 2d of the protocol. For example, assume we know that there is a set of 95% of the peers that have maximum pairwise latency of 100 milliseconds. Then we can set U to be 100 milliseconds provided that we add .05 to f_t.

- *Knowledge of* $\ln n$: In our protocol, we require that every processor knows the quantity $\ln n$. It is not possible for every peer in the DHT to know n exactly because of the dynamic nature of the network. However, the "Simple Fault Tolerant DHT" provides an operation which, with high probability, allows any peer to get a constant factor approximation to $\ln n$ (this operation was first described by Malkhi et al in [16]). The operation takes constant time. Using this approximation causes a (small) constant blowup in the size of the samples chosen in step 2a of our protocol.

- *Ability to Choose a Random Processor*: In our protocol, we assume that each processor can choose a processor uniformly at random. We now present a heuristic for selecting a peer uniformly at random from the set of all current peers in the DHT. We assume that each peer has a unique id (e.g. the IP-address of the peer) and that the domain of the DHT is the interval between 0 and some constant D. Our heuristic simply picks a random number in this interval between 0 and D and then uses the DHT to find the unique peer to which this random number maps[3]. Provided that the the DHT has sufficiently good load-balancing properties, this chosen peer will be a good approximation to a uniformly random choice. Note that since we are assuming a robust DHT, this heuristic allows for

[3]If the number maps to multiple peers, we select one of the peers uniformly at random

sampling even in the presence of Byzantine faults. Using this heuristic, each peer will have $O(\log n)$ latency and send $O(\log^3 n)$ messages to choose the sample set in step 2a of our protcol.

- *Global Coin Toss*: We present a simple heuristic to implement a global coin toss. In the first round of the protocol, each peer selects as leader the peer that is clockwise closest to the point 0 in the DHT. In the i-th round, each peer selects as leader the peer that is clockwise closest to the peer selected in the $i-1$ round. The peer that is selected as leader performs the global coin toss and communicates the results to all the other peers. If we assume that the base hash function of the DHT is a random oracle function [2], then with constant probability, the peer chosen in a given round will be a correct peer. In this case, the global coin toss will be unbiased. The only damage a faulty peer can do when selecting the coin toss is to bias the coin toss or to communicate different values to different peers. The only negative effect this has is to potentially extend the protocol another round. Since no more than a constant fraction of the peers are faulty, this heuristic adds only a constant number of rounds to the protocol in expectation. The heuristic introduces a latency of $O(\log n)$ and requires $O(\log^2 n)$ messages to be sent in step 2f of our protocol(to determine the value of the coin toss).

3.1 A Note on the Global Coin Toss

Our heuristic for the global coin toss works only if a trusted third party assigns IDs to the faulty peers when they join the DHT. Thus, we are assuming a weaker adversary than is normally assumed for the global coin toss problem. In particular, our adversary controls a set of peers which are assigned IDs and then added to the DHT; it can not carefully choose the peers it controls from among those peers already in the network. We feel that our adversary, although somewhat weakened, still covers a broad range of possible attacks. Finding a scalable solution to the global coin toss problem with a more powerful adversary is an area for future work.

4 Proofs

In this section, we give the proof of Theorem 1. Before we give that proof, we first show several lemmas. These lemmas will all be concerned with a fixed round of the protocol.

We will use the following definitions in these lemmas: n is the total number of processors in the protocol; C is the constant in step 2a of the protocol; S_{majority} is the set of correct processors that have the majority bit; S_{faulty} is the set of faulty processors; $M = |S_{\text{majority}}|$; $T = |S_{\text{faulty}}|$; $f_T = T/n$; $f_M = M/n$. For any processor i: S_i is the set of processors in the sample that node i takes in step 2a of the protocol; $t_i = |S_i \bigcap S_{\text{faulty}}|$, $\chi_i = |S_i \bigcap S_{\text{majority}}|$; maj_i, $vote_i$, m_i and M_i are the values computed in the protocol.

The following lemma about sampling will be useful. Intuitively, it says that if the constant C is chosen large enough in step 2a of the protocol, that with high probability, for all processors i, the random variables t_i and χ_i will be close to their expected values.

Lemma 2. *Let α be any fixed constant such that $0 \leq \alpha \leq 1$, Then with probability at least $1 - 3n^{1-2\alpha^2 C}$, the following statements are true for every processor i:*

1. $t_i \leq (f_t + \alpha)C \ln n$
2. $|\chi_i - f_M C \ln n| \leq \alpha C \ln n$

Proof. Let i be some fixed processor. Since each processor in S_i is faulty independently with probability f_t, we know that $E(t_i) = f_t C \ln n$. Further, we can apply Chernoff bounds *todo*, to say that:

$$P(t_i > (f_t + \alpha)C \ln n) \leq e^{-2\alpha^2 C \ln n}$$

Now let ξ_1 be the event that for *any* processor i:

$$t_i > (f_t + \alpha)C \ln n$$

Since the value i can take on only n possible values, we can use a simple Union Bound to bound the probability of event ξ_1:

$$\begin{aligned} P(\xi_1) &\leq ne^{-2\alpha^2 C \ln n} \\ &= e^{(1-2\alpha^2 C)\ln n} \end{aligned}$$

Now again let i be some fixed processor. Since each processor in S_i is from the set S_{majority} independently with probability f_M, we know that $E(\chi_i) = f_M C \ln n$. Further, we can apply Chernoff bounds to say that:

$$P(|\chi_i - f_M C \ln n| > \alpha C \ln n) \leq 2e^{-2\alpha^2 C \ln n}$$

Now let ξ_2 be the event that for *any* processor i:

$$|\chi_i - f_M C \ln n| > \alpha C \ln n$$

Again, since the value i can take on only n possible values, we can use a simple Union Bound to bound the probability of event ξ_2:

$$\begin{aligned} P(\xi_2) &\leq n2e^{-2\alpha^2 C \ln n} \\ &= 2e^{(1-2\alpha^2 C)\ln n} \end{aligned}$$

Finally we can use a simple Union bound to bound the probability that either event ξ_1 or event ξ_2 occur:

$$\begin{aligned} P(\xi_1 \cup \xi_2) &\leq 3e^{(1-2\alpha^2 C)\ln n} \\ &\leq 3n^{1-2\alpha^2 C} \end{aligned}$$

So for C chosen sufficiently large, the statements in the lemma hold with high probability. □

The following lemmas are all true with probability at least $1 - 3n^{1-2\alpha^2 C}$.

Lemma 3. *Consider some fixed round of the protocol. With high probability, for all processors i:*

$$M - \alpha n \leq M_i \leq M + T + 2\alpha n$$

Proof. To get the upperbound, we first note that for any processor i, $m_i \leq \chi_i + t_i$ and apply Lemma 2 to upperbound χ_i and t_i:

$$\begin{aligned} m_i &\leq \chi_i + t_i \\ &\leq (f_M + \alpha)C \ln n + (f_T + \alpha)C \ln n \\ &= (f_M + f_T + 2\alpha)C \ln n \end{aligned}$$

This implies that:

$$M_i \leq M + T + 2\alpha n.$$

To get the lowerbound, we note that for any processor i, $m_i \geq \chi_i$ and then apply Lemma 2 to lowerbound χ_i:

$$\begin{aligned} m_i &\geq \chi_i \\ &\geq (f_M - \alpha)C \ln n \end{aligned}$$

This implies that $M_i \geq M - \alpha n$. □

Corollary 4. *With high probability, for any two processors i and j, $|M_i - M_j| \leq T + 3\alpha n$*

Proof. Without loss of generality, assume that $M_i \geq M_j$. Then from Lemma 3, we know that with high probability, $M_i \leq M + T + 2\alpha n$. This implies that $M \geq M_i - T - 2\alpha n$

Using Lemma 3 again, we can say that:

$$\begin{aligned} M_j &\geq M - \alpha n \\ &\geq (M_i - T - 2\alpha n) - \alpha n \\ &= M_i - (T + 3\alpha n) \end{aligned}$$

□

Lemma 5. *If any processor j, has $M_j \geq H$ in some round, then for all processors i, $maj_i = maj_j$ in that round.*

Proof. Since $M_j \geq H$, we know by Lemma 3 that $M \geq H - (T + 2\alpha)n$. Thus, $f_M \geq (1 - 3f_T - 6\alpha)$. Hence by Lemma 2, for all processors i,

$$\begin{aligned} \chi_i &\geq f_M C \ln n - \alpha C \ln n \\ &\geq (1 - 3f_T - 7\alpha) C \ln n \\ &\geq (1/2) C \ln n \end{aligned}$$

This implies that for every processor i, at least half of S_i is from S_{majority}. This implies that all processors have the same maj value[4] and that in particular, for all processors i, $maj_i = maj_j$. □

Lemma 6. *If all correct processors have the same vote value at the start of some round, then with high probability they will all permanently decide on that value at the end of the round.*

Proof. Let b be the *vote* value that all processors have at the beginning of the round. We first show that for all processors i, $maj_i = b$. This is equivalent to showing that for all processors i, $\chi_i > (1/2) C \ln n$. We know that $f_M = 1 - f_T$, thus by Lemma 2, we know that for all processors i:

$$\begin{aligned} \chi_i &\geq f_M C \ln n - \alpha C \ln n \\ &\geq (1 - f_T - \alpha) C \ln n \\ &> (1/2) C \ln n \end{aligned}$$

Where the last line follows because our constraint $3f_t + 7\alpha = 1/2$ implies that $f_t + \alpha < 1/2$. Thus for all processors i, $maj_i = b$

We next show that all processors have $M_i \geq G$. If all correct processors have the same vote value at the start of some round, then $M = n - T$. Thus, from Lemma 3, we know that with high probability, for all processors i:

$$\begin{aligned} M_i &\geq M - \alpha n \\ &= n - T - \alpha n \\ &= G \end{aligned}$$

□

[4] provided that $C \ln n$ is an odd number, which is easy to ensure.

Lemma 7. *If any processor permanently sets its output value in a given round, then by the end of the following round all processors will have permanently set their outputs to the same value.*

Proof. Let j be some processor which has permanently set its output in the current round to the value d_j. Since $M_j \geq G$, we know by Lemma 5 that for all other processors, i, $maj_i = d_j$. From Corollary 4, we also know that since $M_j \geq G$, for all processors i, $M_i \geq G - (T + 3\alpha n)$ i.e. $M_i \geq H$.

Thus, with high probability, for all processors i, $vote_i = d_j$ at the end of the round. Hence, by Lemma 6, in the next round, all processor's will permanently set their values to d_j. □

Lemma 8. *Any given round will be the last or next to last round with probability at least* $1/2$

Proof. There are two cases for a round.
Case 1: For all processors i, $M_i < H$. In this case, if the global coin toss sets the threshold to be H, all processors will set their vote values to 0 and so the protocol will terminate in the next round by Lemma 6. This happens with probability $1/2$.
Case 2: For some processor j, $M_j \geq H$. By Lemma 5, all processors, i, will have $maj_i = maj_j$ for this round. By Corollary 4, all processors i will have $M_i \geq L$ for this round. Thus if the global coin toss sets the threshold to L, all processors will set their *vote* values to maj_j and so the protocol will terminate in the next round by Lemma 6. This happens with probability $1/2$. □

4.1 Proof of Theorem 1

We are now ready to prove Theorem 1.

Proof. The protocol requires that $1 - 3f_T - 7\alpha \leq 1/2$ for any constant $\alpha > 0$. This implies that f_T must be less than $1/6$ and that $\alpha = (1/14 - 3/7 f_T)$. Call a round *good* if the statements in Lemma 2 (and thus all the lemmas) hold and call it *bad* otherwise. Let P_b be the probability that a given round is bad. Thus:

$$P_b \leq 3n^{1-2\alpha^2 C}$$

Let ξ_i be the event that the i-th round is the first bad round. Note that $P(\xi_i)$ is no more than the probability that the protocol gets to the i-th round times P_b. Thus, $P(\xi_1) \leq P_b$, and for all $i \geq 2$ $P(\xi_i) \leq (1/2)^{i-1} P_b$. Now the probability that *any* round is bad is $P(\bigcup_{i=1}^{\infty} \xi_i)$ which we can bound with a Union bound as follows:

$$\begin{aligned} P(\bigcup_{i=1}^{\infty} \xi_i) &\leq \sum_{i=1}^{\infty} P(\xi_i) \\ &\leq P_b + \sum_{i=0}^{\infty} (1/2)^i P_b \\ &= 3P_b \\ &\leq 9n^{1-2\alpha^2 C} \\ &\leq 9n^{1-2(\frac{1}{14} - \frac{3}{7} f_T)^2 C} \end{aligned}$$

We note that if no round is bad, we can establish both correctness and termination. Correctness is established by Lemmas 6 and 7: Lemma 6 proves that if all correct processors have the same input, then they will all have the same output. Lemma 7 proves that all correct processors will have the same output result. Finally, termination of the protocol is established by Lemma 8. In particular, the expected number of rounds is no more than 3. We note that each processor sends $2C \ln n$ expected messages per round ($C \ln n$ messages asking for bits and $C \ln n$ messages sending the processor's vote). Thus, the expected total number of messages sent and received by a processor in the protocol is no more than $6C \ln n$. □

5 Conclusion and Open Problems

We have presented a first result on the scalable Byzantine agreement problem. Numerous open problems remain including the following:

- Can one design a scalable solution to the global coin toss problem for an adversary which is able to carefully choose the peers it controls from among all the peers in the network?
- In many cases, it may suffice to guarantee that, e.g., only 99% of the processors reach agreement. In such a situation, is it possible to reduce the message complexity to $O(1)$ in expectation?
- Can one apply the same type of sampling technique used in this paper to other classical problems in distributed computing in order to make them more scalable?

6 Acknowledgements

We gratefully thank James Aspnes, John Douceur and Stefan Saroiu for their help with this paper.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[2] 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.

[3] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 27–30, Montreal, Quebec, Canada, 17–19 August 1983.

[4] P. Berman and J. Garay. Randomized distributed agreement revisited. In *Proc. 23rd International Symposium on Fault-Tolerant Computing*, 1993.

[5] Kenneth P. Birman. The surprising power of epidemic communication. In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 97–102. Springer, 2003.

[6] John Borland. Gnutella girds against spam attacks. *CNET News.com*, August 2000. http://news.cnet.com/news/0-1005-200-2489605.html.

[7] John Borland. Hackers, madonna mix it up. *CNET News.com*, April 2003. http://news.com.com/2100-1025-997856.html.

[8] G. Bracha. An asynchronous [(n-1/3)]-resilient consensus protocol. In *ACM Conference on the Principles of Distributed Computing(PODC)*, 1984.

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

[10] John Douceur. The sybil attack. In *Proceedings of the Second Internation Peer to Peer Symposium (IPTPS)*, 2002.

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

[12] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[13] Verisign Incorporated. http://www.verisign.com.

[14] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[15] K.R. Lefebvre. "the added value of embassy in the digital world". Technical report, Wave Systems Corporation, 2000.

[16] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic lookup network. In *ACM Conference on the Principles of Distributed Computing(PODC*, 2002.

[17] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[18] M. Naor and U. Wieder. A simple fault tolerant distributed hash table, 2003.

[19] M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[20] M. O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (FOCS '83)*, pages 403–409, Los Alamitos, Ca., USA, November 1982. IEEE Computer Society Press.

[21] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001.

[22] John Spooner and Ken Popovich. Intel: The future is peer. *ZDNet News*, August 2000. http://zdnet.com.com/2100-11-523296.html?legacy=zdnn.

[23] I. Stoica, D. Atkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of the ACM SIGCOMM 2002 Technical Conference*, 2002.

[24] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001.

[25] S. Toueg. Randomized byzantine agreements. In *ACM Conference on the Principles of Distributed Computing(PODC)*, 1984.

[26] DataSynapse Website. http://www.datasynapse.com.

[27] FOLDING@home Website. http://folding.stanford.edu.

[28] Gnutella Website. http://gnutella.wego.com/.

[29] Groove Networks Website. http://www.groove.net.

[30] Kazaa Website. http://www.kazaa.com.

[31] Morpheus Website. http://gnutella.wego.com/.

[32] Napster Website. http://www.napster.com/.

[33] SETI@home Website. http://setiathome.ssl.berkeley.edu.

[34] B.Y. Zhao, K.D. Kubiatowicz, and A.D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley Technical Report, April 2001.

[35] Andrew Zolli. Monsters of rock: Meet the music industry agents that could ruin your downloading career. *Wired Magazine*, September 2003. www.wired.com/wired/archive/11.09/start.html?pg=12.