Scalable Byzantine agreement with a random beacon

Olumuyiwa Oluwasanmi and Jared Saia

{muyiwa,saia}@cs.unm.edu; Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386. This research was partially supported by NSF CAREER Award 0644058, NSF CCR-0313160, and an AFOSR MURI grant.

Abstract. We present two Monte Carlo algorithms for efficiently computing Byzantine agreement in the partially synchronous communication model. The algorithms assume the existence of a random beacon, which is a stream of random bits known to all processors. Both algorithms terminate in $O(1)$ expected time. The first algorithm sends $O(B + n \log n)$ messages in total, where B is the maximum number of messages sent by the bad processors in any round and n is the number of processors. It ensures all processors reach agreement. The second algorithm sends $\tilde{O}(1)$ messages per processor, and is thus load-balanced, and ensures all but a $o(1)$ fraction of the processors reach agreement. Both algorithms succeed with probability $1 - O(1/n^k)$, even against an adaptive adversary that takes over up to a $1/3 - \epsilon$ fraction of the processors, for any fixed positive values ϵ and k. We prove the correctness of both algorithms and provide empirical evidence that they require significantly less bandwidth than state of the art for networks of size greater than 4,000 processors. Our algorithms work in the full-information model and thus make no cryptographic assumptions.

1 Introduction

Peer-to-peer networks are not a fad. The idea of a decentralized network, in which each processor plays the role of both a client and a server, is now an important technique in our toolkit for designing Internet-scale applications. The peer-to-peer approach continues to be used in established and thriving applications such as Bittorrent [24] and Skype [25], and the peer-to-peer approach even now still finds new uses in applications such as Bitcoin [23].

Peer-to-peer networks offer the promise of improved reliability when compared to traditional client-server networks. However, such promise will never be achieved *unless we can design a network that is more reliable than any of its individual components*. Unfortunately, most peer-to-peer algorithms are easily defeated by an adversary that controls even a single processor.

In this paper, we address this problem by designing scalable algorithms that solve Byzantine agreement. In the Byzantine agreement problem, there are n processors some of which are *bad*, in that they are controlled by an adversary that can cause them to deviate in an arbitrary way. The remaining processors are *good* in that they follow the protocol. The goal is to ensure all good processors output the same bit, and that this bit equals the input bit of some good processor. Intuitively, a solution to Byzantine agreement allows us to build a reliable system from unreliable components.

We take a step towards creating Byzantine agreement algorithms that scale to large networks, like peer-to-peer networks. Our algorithms are *scalable* in the sense that they do not require all-to-all communication. Critical to our approach is the utilization of a random beacon. A random beacon is a random stream of bits available to all the processors. It was first defined by Rabin in [21] where it was used to build a contract signing protocol. Maurer et al. in [19,5,2,8] describe a computation model called the Memory Limited or Limited Storage space Model, which uses the existence of a random beacon, to build a secure communication system based on private keys.

1.1 Model

Our algorithms assume that n processors can communicate directly via point-to-point message passing, and that the ID's of all processors are globally known. Our communication model is *partially synchronous*. In particular, we assume a known upper bound on the time it takes to receive a message. However, we do not assume that processors run in lock-step, and we can tolerate a *rushing adversary* that gets to read all the messages from good processors before sending the messages from bad processors in a round.

We assume the *full-information model*, meaning that all messages sent are public and can be seen by the bad processors; we thus make no cryptographic assumptions. We assume t processors are controlled by an adaptive adversary where $t \leq (1/3 - \epsilon)n$ for some fixed $\epsilon > 0$. The adversary is adaptive in the sense that it can take over processors at any point in the protocol up to taking over t processors and once processors are corrupted, they stay corrupted [18]. We also assume that the bits from the random beacon are only available at the beginning of a round.

1.2 Our Results

This paper describes two algorithms, RBQUERY and RBSAMPLER that can solve Byzantine agreement without all-to-all communication. To the best of our knowledge, this paper presents the first *practical* algorithms with this property. Dolev and Reischuk [7] showed that any algorithm must send $\Omega(n^2)$ messages to ensure Byzantine agreement with probability 1. This lower bound still holds even with the assumption of a random beacon. Thus, our algorithms are necessarily Monte Carlo, in the sense that they succeed with probability $1 - O(1/n^k)$, for any fixed k.

Our main results are given in the following two theorems. For the first theorem, we let B be an upper bound on the total number of messages sent in a round by the bad processors. We note that B is at most $\theta(n^2)$, but may be much lower, depending on the resources available to the adversary.[1] RBQUERY is described in Section 2.1, and the proof of Theorem 1 is given in Section 3.1.

Theorem 1. *For any positive k, there exist settings of parameters C and ϵ_0, in the algorithm, to ensure the following with probability at least $1-1/n^k$.* RBQUERY *is correct, in that each good processor terminates with the same value and this value equals the input bit of some good processor, and moreover:*

[1] We assume that the bad processors can communicate amongst themselves for free, so B counts only messages that bad processors send to good processors.

– *All good processors terminate in $O(\log n)$ rounds; and*
– *All good processors terminate in $O(1)$ rounds in expectation; and*
– *The total number of bits sent by all good processors is $O(B+n\log n)$ in expectation.*

Our next algorithm is called RBSAMPLER. This algorithm make use of a sampler in determining the communication graph among the processors. RBSAMPLER is thus non-uniform in the sense that there is a different version of the algorithm for each value of n. Also, unlike the RBQUERY, RBSAMPLER only achieves agreement that is *almost-everywhere* in the sense that all but a $o(1)$ fraction of the processors reach agreement. RBSAMPLER is described formally in Section 2.2 and the proof of the following theorem is presented in Section 3.2.

Theorem 2. *For any positive k and d, there exist settings of parameters C and ϵ_0, in the algorithm, to ensure the following with probability at least $1 - 1/n^k$, for* almost all *processors i.e. all but $O(\log^{d-1} n)$.* RBSAMPLER *is correct, in that almost all processors terminate with the same value and this value equals the input bit of some good processor; and moreover:*

– *Almost all processors terminate in $O(\log n)$ rounds; and*
– *Almost all processors terminate in $O(1)$ rounds in expectation; and*
– *Almost all processors send polylog(n) bits.*

1.3 Our Approach

Our algorithms build on recent work by King and Saia[12] that describe Byzantine agreement algorithms that have $\theta(\log n)$ latency and require each processor to send $\tilde{O}(n^{3/2})$ messages. In contrast to [12], our algorithms assume the existence of a random beacon. However, our algorithms improve on this previous work in the following ways: 1) we achieve an expected latency of $O(1)$; 2) we tolerate an adaptive adversary, while the work in [12] tolerates only a non-adaptive adversary; 3) our algorithms send significantly fewer messages in practice; and 4) our algorithms are significantly simpler: the algorithm in [12] is several pages of detailed pseudocode, while our algorithms are each described in half a page.

Like many Byzantine agreement algorithms, our algorithms follow the basic structure of Rabin's algorithm [21]: each processor starts with an individual vote; votes are exchanged with other processors in each round; and finally each processor updates their vote for the next round based on messages received. However, to achieve reduced communication costs, the processors in our algorithm are only able to talk to a small sample of all the processors in the network. The challenging problem is achieving Byzantine agreement, while using only the information contained in these small samples.

The technically challenging part of this paper is the design of the algorithm RBQUERY, and the proof of Theorem 1, which directly addresses how to achieve agreement while talking only to small samples of processors. Once we have established this algorithm and theorem, it is relatively straightforward to design RBSAMPLER and prove Theorem 2. In particular, Theorem 2 essentially follows by combining Theorem 1 with a result on samplers from [13].

1.4 Related work

The Byzantine agreement problem was first proposed by Lamport et al. [16] as a way to model basic fault-tolerant distributed computing. Lamport, Shostak and Pease proved in [16] that Byzantine agreement is only possible when less than $n/3$ of the processors are bad. Fischer, Lynch and Paterson in [11] also proved that a single Byzantine fault in the asynchronous model of communication renders Byzantine agreement impossible for a deterministic algorithm.

Byzantine agreement is used in peer-to-peer systems such as Farsite [1] and Oceanstore [15,22]. It is also a critical part of proposed systems for many large wireless and mobile networks. See [14] for a comprehensive discussion of why Byzantine agreement is important for large-scale peer-to-peer networks.

Rabin in [20] solved a randomized version of the Byzantine agreement problem using the notion of a *common coin* (this is what we refer to as a random beacon). A common coin is a value that is only available to all the processors at the beginning of each round of computation. It is 0 with probability $1/2$ and 1 with probability $1/2$. Rabin's algorithm [20] takes $O(1)$ expected rounds and requires $\Theta(n^2)$ messages. Rabin's algorithm works in the asynchronous, full information model and is correct with probability 1.

In the empirical section of this paper, we compare the resource costs of our algorithm with a popular Byzantine agreement algorithm proposed by Cachin, Kursawe and Shoup [4], which we refer to as the CKS algorithm. The CKS algorithm tolerates up to $n/3$ bad processors, runs in constant expected time, and sends $\Theta(n^2)$ messages. It does not assume a random beacon. However, unlike our algorithms, CKS does require the establishment of a public key infrastructure.

We choose CKS as a point of comparison, because, in practice, the number of messages it requires processors to send is the smallest among prior Byzantine agreement. In particular, in our experiments, CKS generally required only about $4n^2$ messages, which is better than other algorithms of which we are aware.

1.4.1 Related Work: Random Beacon

Lee, Clark et al. showed in [17,6] how to build a random beacon using publicly available information on the Internet, specifically using information from the Dow Jones Industrial Average (DJIA) or from other financial data. Eastlake in [9,10] also describes how to build a selection protocol out of a random beacon.

A concern is the actual amount of randomness present in implementations of a random beacon from publicly available information, this could be a problem if the publicly available data could be manipulated in some way and hence introducing some bias into the random beacon bits. In particular, Clark and Hengartner in [6] determined via experiments the entropy of the stock components of the Dow Jones Industrial Average (DJIA) and show that the Shannon entropy for a single stock in the DJIA per day is between 6.83 and 9.45. They create a random beacon from the DJIA and show a Shannon entropy of 218 bits for their random beacon.

We further note that hardware implementations of a random beacon have been proposed [21]. These may include: a satellite broadcasting random bits, or a trusted node in a massively parallel computer or wireless network.

1.5 Paper Layout

The rest of this paper is organized as follows. In Section 2, we formally describe our algorithms: RBQUERY in described in Section 2.1, and RBSAMPLER in Section 2.2. In Section 3, we analyze our algorithms: the proof of Theorem 1 is given in Section 3.1, and the proof of Theorem 2 is given in Section 3.2. In Section 4, we empirically test our algorithms via simulations for networks containing up to a million nodes. Finally, in Section 5, we conclude and give directions for future work.

2 Algorithms and their Description

2.1 RBQUERY

Our first algorithm RBQUERY is presented as Algorithm 2.1 (RBQUERY). This algorithm has two parameters: C and ϵ_0. Both of these values depend on k and ϵ as is described formally in the proof of Lemma 1.[2] In our experiments, we set $C = 40$ and $\epsilon_0 = 1/8$.

We now describe RBQUERY at a high level. In each round, each processor has a vote which is the bit held by that processor at the start of the round. Also in each round, a processor selects $\Theta(\ln n)$ processors uniformly at random with replacement to query for their votes. Each processor sends its vote to the processors that queried it, while it receives votes from the processors that it queried. After each round, the fraction of votes received that are for the majority bit is computed.

If the fraction of processors that vote for the majority bit is at least $(1-\epsilon_0)(2/3+\epsilon/2)$, then the processor sets its vote to the majority bit. Otherwise the processor sets its bit to the value from the random beacon. A processor terminates after it has 1) computed a fraction value at least $(1-\epsilon_0)(2/3+\epsilon/2)$ with a majority bit value b; and 2) the global coin value equals b twice.

Note that if all processors simply commit to the random beacon, then we would not be able to solve Byzantine agreement, as the bit from the random beacon may not be the input bit of any good processor. Such a scenario violates the conditions for Byzantine agreement.

We analyze the correctness of RBQUERY in Section 3.1.

2.2 RBSAMPLER

We now describe the algorithm RBSAMPLER. The main difference in this algorithm, compared to RBQUERY, is that it makes use of a sampler in determining the communication graph among the processors. RBSAMPLER is thus non-uniform in the sense that there is a different version of the algorithm for each value of n. Also, the neighbors of each processor are fixed and do not change between rounds, unlike RBQUERY. Finally, unlike the RBQUERY, RBSAMPLER only achieves almost-everywhere agreement.

We analyze the correctness of RBSAMPLER in Section 3.2.

[2] Recall $1/n^k$ is the probability of failure and the fraction of bad nodes is no more than $1/3-\epsilon$.

Initialize:

1. $vote \leftarrow b_i$
2. $Match \leftarrow FALSE$

Repeat until termination:

1. Select $C \ln n$ processors uniformly at random with replacement. Set $InNeighbors$ to these processors. Send request messages to all processors in $InNeighbors$
2. Receive all request messages. Set $OutNeighbors$ to all processors from which request messages were received
3. Send vote to all processors in $OutNeighbors$
4. Collect all votes received from processors in $InNeighbors$
5. $coin \leftarrow$ next output of random beacon
6. If $Match$ then
   (a) If $coin = vote$ then commit to value $vote$ and **terminate**
7. Else
   (a) $maj \leftarrow$ majority vote among $InNeighbors$
   (b) fraction $\leftarrow$ fraction of votes received for maj
   (c) If $fraction \geq (1 - \epsilon_0)(2/3 + \epsilon/2)$ then
      i. $vote \leftarrow maj$
      ii. If $coin = vote$ then $Match \leftarrow TRUE$
   (d) else
      i. $vote \leftarrow coin$

Algorithm 2.1: RBQUERY

3 Analysis and proofs

In this section, we show the correctness of RBQUERY and RBSAMPLER by proving Theorem 1 and Theorem 2.

3.1 Analysis and proofs for RBQUERY

We begin with a proof of correctness for RBQUERY. For a fixed round, let $b \in \{0, 1\}$ be the bit that the majority of good processors vote for in that round (breaking ties arbitrarily). Let M be the set of good processors that will vote for b, and let $f = |M|/n$. Let $0 \leq \epsilon_0 \leq 1$ be a fixed constant to be determined later. We call a processor *informed* for the round if the fraction value for that processor obeys the following inequalities:

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

Lemma 1. *For any positive integers C' and ℓ, there exists setting of parameters C and ϵ_0 in Algorithm 2.1 (*RBQUERY*), such that the following holds. Let ℓ' be the minimum of ℓ and r, where r is the first round in which a good processor terminates. Then all good processors are informed for the first ℓ' rounds with probability at least $1 - 2\ell'/n^{C'}$.*

Proof. Fix a round $r' \leq \min(\ell, r)$. Let G be a bipartite multigraph induced by the $InNeighbors$ and $OutNeighbors$ selection process defined in RBQUERY for round r'.

Initialize:

1. $vote \leftarrow b_i$
2. $Match \leftarrow FALSE$
3. Set $InNeighbors$ (resp. $OutNeighbors$) to all processors that have in-edges (resp. out-edges) to this processor in the sampler.

Repeat until termination:

1. Send vote to all processors in $OutNeighbors$
2. Collect votes from all processors in $InNeighbors$
3. $coin \leftarrow$ next output of random beacon
4. If $Match$ then
   (a) If $coin = vote$ then commit to value $vote$ and **terminate**
5. Else
   (a) $maj \leftarrow$ majority bit among $InNeighbors$
   (b) $fraction \leftarrow$ fraction of votes received for maj
   (c) If $fraction \geq (1 - \epsilon_0)(2/3 + \epsilon/2)$ then
      i. $vote \leftarrow maj$
      ii. If $coin = vote$ then $Match \leftarrow TRUE$
   (d) else
      i. $vote \leftarrow coin$

Algorithm 2.2: RBSAMPLER

G is constructed as follows. There are n nodes on the left hand side and copies of all these nodes on the right hand side. The adversary chooses a subset of $1/3 - \epsilon$ of the nodes on the left that are bad. Edges are created as follows: each good node on the right chooses $C \log n$ neighbors on the left uniformly at random with replacement, and forms edges with these nodes.

Let b be the bit that the majority of good processors vote for in round r' and let M be the set of good processors that vote for b. We know that M is of size at least $(1/3+\epsilon/2)n$ since at least half of the good processors must vote for the majority bit. Let $f = |M|/n$. Note that the edges in G are chosen independently of the set M.

Fix a good node, p, on the right hand side of G. Let X be the number of edges from p into the set M. Note that $E(X) = fC \ln n$. Moreover, each edge from p falls into some processor in M independently with probability f. Thus, we can apply Chernoff bounds to say that for any positive ϵ_0,

$$Pr(X < (1 - \epsilon_0)E[X]) < e^{-E[X]\epsilon_0^2/2} = e^{-(fC\epsilon_0^2/4)\ln n}.$$

Similarly, we can use Chernoff bounds to say that

$$Pr(X > (1 + \epsilon_0)E[X]) < e^{-E[X]\epsilon_0^2/3} = e^{-(fC\epsilon_0^2/6)\ln n}.$$

Hence the probability that either of these bounds is violated is no greater than the sum of these two probabilities, or less than $2e^{-(fC\epsilon_0^2/6)\ln n}$. Clearly if both of these bounds hold, then the fixed good processor is informed for this round.

Now let $\xi_{p,r'}$ be the event that either of these bounds is violated for some fixed processor p in some fixed round r'. Further let ξ be the event that either of these bounds is violated for any good processor p in any of the first ℓ' rounds. Then by a union bound, we have that

$$\begin{aligned} Pr(\xi) &= \sum_{p,r'} Pr(\xi_{p,r}) \\ &\leq (n\ell')2e^{-(fC\epsilon_0^2/6)\ln n} \\ &\leq 2\ell'/n^{C'} \end{aligned}$$

Note that $f \geq 1/3$, so the last equation holds provided that C is sufficiently large, but depending only on ϵ_0 and C'.

Lemma 2. *Assume 1) all good processors have vote value equal to b at the beginning of some round r; 2) no good processor has terminated prior to round r; and 3) all processors are informed in round r. Then all good processors will have vote value equal to b at the beginning of round $r+1$.*

Proof. If all good processors have vote value equal to b at the beginning of round r, it means that $f = |M|/n = 2/3+\epsilon$. Since all processors are informed in round r, it means that for each processor, $fraction \geq (1-\epsilon_0)f \geq (1-\epsilon_0)(2/3+\epsilon)$. Thus, each processor at the end of round r will set its vote to the majority value, which equals b, and hence will have vote value equal to b at the beginning of round $r+1$.

Lemma 3. *Let r be a positive integer and $\epsilon_0 < (3/4)\epsilon$. Assume that 1) all processors are informed in round r; and 2) that no good processor has terminated prior to round r. Let S_h be the set of good processors in round r that have $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$. Then, at the end of round r, all processors in S_h will have the same vote value.*

Proof. We show this by contradiction. Assume there are two processors, x and y, where $fraction_x$ ($fraction_y$) are the fraction values of x (y, resp.), such that both $fraction_x$ and $fraction_y$ are greater than or equal to $(1-\epsilon_0)(2/3+\epsilon/2)$. Assume further that x sets its vote to 0 at the end of the round, while y sets its vote to 1.

Let f_0 be the fraction of good processors that vote for 0 during round r; and let f_1 be the fraction of good processors that vote for 1 during round r. By assumption, $fraction_x \geq (1-\epsilon_0)(2/3+\epsilon/2)$. By the definition of informed, we also know that $fraction_x \leq (1+\epsilon_0)(f_0+1/3-\epsilon)$. This implies that

$$(1-\epsilon_0)(2/3+\epsilon/2) \leq (1+\epsilon_0)(f_0+1/3-\epsilon).$$

Isolating f_0 in this inequality, we get that

$$f_0 \geq \frac{1/3+(3/2)\epsilon-\epsilon_0+(1/2)\epsilon\epsilon_0}{1+\epsilon_0}.$$

A similar analysis for $fraction_y$ implies that

$$f_1 \geq \frac{1/3+(3/2)\epsilon-\epsilon_0+(1/2)\epsilon\epsilon_0}{1+\epsilon_0}.$$

But then we have that,

$$\begin{aligned} f_0 + f_1 &\geq \frac{2/3 + 3\epsilon - 2\epsilon_0 + \epsilon\epsilon_0}{1 + \epsilon_0} \\ &> 2/3 + \epsilon \end{aligned}$$

where the last line is clearly a contradiction (holding if $\epsilon_0 < (3/4)\epsilon$).

Lemma 4. *Assume $\epsilon_0 < (3/4)\epsilon$; let r be the first round in which a good processor terminates; and assume all processors are informed in rounds 1 to r. Further, assume that some good processor commits to value b in round r. Then 1) all good processors that commit in round r will commit to b; and 2) in round $r + 1$, all remaining good processors will set vote value equal to b and $Match$ value to $TRUE$. Hence, all good processors will have committed to b by the next round in which $coin = b$.*

Proof. Consider a processor p that commits to bit value b in round r. Then processor p's $Match$ value must be equal to $TRUE$ in round r. This means there must have been some previous round, r', in which p's $Match$ value was first set to $TRUE$. Among processors that commit to values in round r, let p be a processor with the smallest such r' value. Note that at the point that $Match$ was set to $TRUE$ in round r', p's vote value must have been b, since a processor's vote value can not change after its $Match$ value is set to true.

Let S_h be the set of good processors in round r' that have $fraction \geq (1-\epsilon_0)(2/3+\epsilon/2)$. By Lemma 3, all processors in S_h set their vote value to b at the end of the round. Now since processor p set $Match$ to $TRUE$ in round r', it must be the case that the outcome of the global coin in that round was equal to b. This means at the end of round r', *all* good processors set their vote values to b.

But then, by then by repeated applications of Lemma 2, for all rounds subsequent to round r', all processors will have vote values equal to b and so if they commit to any value, it will be the value b. Further, note that for p to terminate, in round r, it must be that $coin = b$. Thus, all remaining good processors will set their $Match$ values to $TRUE$ at the beginning of round r.

Lemma 5. *Let $\epsilon_0 < (3/4)\epsilon$ and r be a positive integer. Assume that all good processors are informed in round r; and that no good processor has terminated prior to round r. Then, with probability at least $1/2$, at the end of round r, all good processors will have the same vote value.*

Proof. Let S_h be the set of good processors in round r that have $fraction \geq (1 - \epsilon_0)(2/3 + \epsilon/2)$. By Lemma 3, all processors in S_h will set their vote value to the same value, call it b, at the end of the round. But with probability $1/2$, the global coin in round r will have value b, and so all the remaining good processors will also set their vote value to b.

We now prove Theorem 1.

Proof. Throughout this proof, we will assume that ϵ_0 is set such that $\epsilon_0 < (3/4)\epsilon$.

We first show that, with high probability[3], all good processors terminate in $O(\log n)$ rounds; and that all good processors terminate in $O(1)$ rounds in expectation. Fix a constant C_1 and let $\ell = C_1 \log n$; let r' be the first round in which a good processor terminates; and let $\ell' = \min(\ell, r')$. By Lemma 1, for any constants C', there exist settings of the algorithm parameters such that all good processors will be informed for ℓ' rounds with probability at least $1 - 2C_1 \log n / n^{C'}$. By Lemma 4, if r' is the first round in which a good processor terminates and commits to value b, then in round $r' + 1$, all processor will set their $Match$ value to $TRUE$ and so will terminate the next time $coin = b$. Thus, the number of remaining rounds after r' until all good processors terminate is bounded by a geometric random variable with success probability $1/2$.

We now consider the distribution of r'. By Lemma 5, in each of the first ℓ' informed rounds, there is probability at least $1/2$ that at the end of that round, all good processors will have the same vote value. Moreover, by Lemma 2, once all good processors have the same vote value in some round, they will all continue to have the same vote value, as long as 1) they are all informed; and 2) none of them terminate. Thus, three events need to occur for us to reach round r': 1) all processors first obtain the same vote value, b (this happens with probability $1/2$ in any round); and 2) the value $coin$ equals b two times (the event $coin = b$ happens with probability $1/2$ in any round).

Thus, the distribution of r' is a random variable that is upper bounded by the sum of two geometric random variables with success probability $1/2$. Hence, the total number of rounds until all processors terminate is upper bounded by the sum of four geometric random variables with probability $1/2$. Clearly the expected value of such a summation is $O(1)$ and, with high probability, the sum is no more than $C_1 \log n$.

We now argue correctness. First, by Lemma 2, if all good processors initially have input values equal to b, then they will all always have vote values b, so they will all eventually commit to b. Next, Lemma 4 shows that if any good processor commits to some value b, that all good processors will eventually commit to b.

Finally, we count the expected number of bits sent. In any round, a good processor sends out messages to $\theta(\log n)$ other processors and the bad processors send out a total of at most B messages. Thus, the total number of requests received by the good processors is $\theta(B + n \log n)$. To answer requests thus requires sending a total of $\theta(B + n \log n)$ messages, each containing a single bit. Since the expected number of rounds is $O(1)$, the expected number of bits sent in total is $\theta(B + n \log n)$ by linearity of expectation.

3.2 Analysis and proofs for RBSAMPLER

For our new algorithm, we use the same definition as in Section 3.1 to say whether a processor is informed in a given round. The proof of the following lemma is equivalent to that in King and Saia [13].

Lemma 6 (Graph existence). *For any positive k and positive constants ϵ_0 and d, there exists a directed multigraph G on k vertices with maximum out-degree no more than $C_3 \log^d n$, where C_3 depends only on ϵ_0 and d, such that if this communication graph is*

[3] We will use the term "with high probability" to mean that there is probability of error no more than $1/n^k$ for a constant k.

*used in Algorithm 2.2 (*RBSAMPLER*), then in every round all but a* $1/\log^d n$ *fraction of the good processors are informed.*

The following simple corollary follows by summing up the number of processors that are not informed in every round.

Corollary 1. *Let* C' *be a positive integer. If the conditions of Lemma 1 are met then all but a* $C'/\log^{d-1} n$ *fraction of the good processors are informed in every one of the first* $C' \log n$ *rounds.*

We will say that a processor is *always informed* if it is informed for every round from the start of Algorithm 2.2 (RBQUERY), to the round in which the processor terminates.

The proof of Theorem 2 now follows in the same manner as the proof of Theorem 1, with the following changes. First, we use Corollary 1 in place of Lemma 1. Second, up to a $O(1/log^{d-1}n)$ fraction of the good processors are not informed in all rounds and so may not terminate or may not terminate with the correct value. Finally, all requests are only answered from neighbors in the sampler, so the total number of bits sent in expectation is now $O(\log^d n)$.

4 Experimental Results

In this section, we describe the results of simulating RBQUERY and RBSAMPLER on networks containing from 1,000 to 1,024,000 nodes. We first describe the experimental setup and then describe the outcomes.

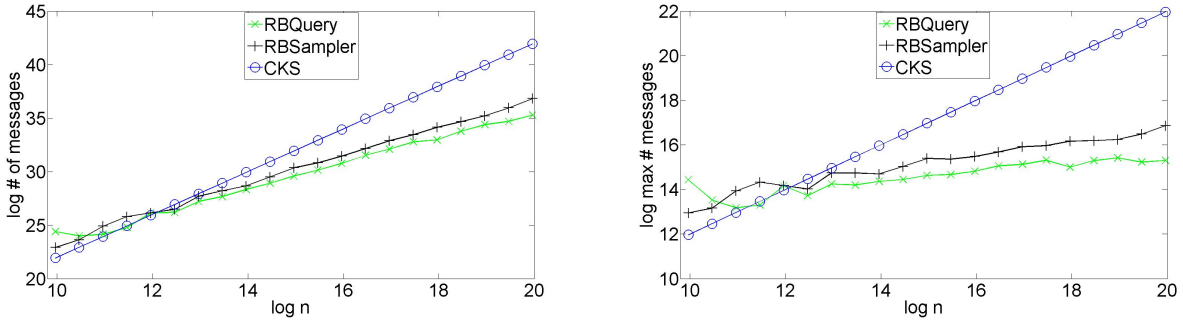

Fig. 1. Left: Log of number of nodes vs. number of messages sent; Right: Log of number of nodes vs max messages sent by a node.

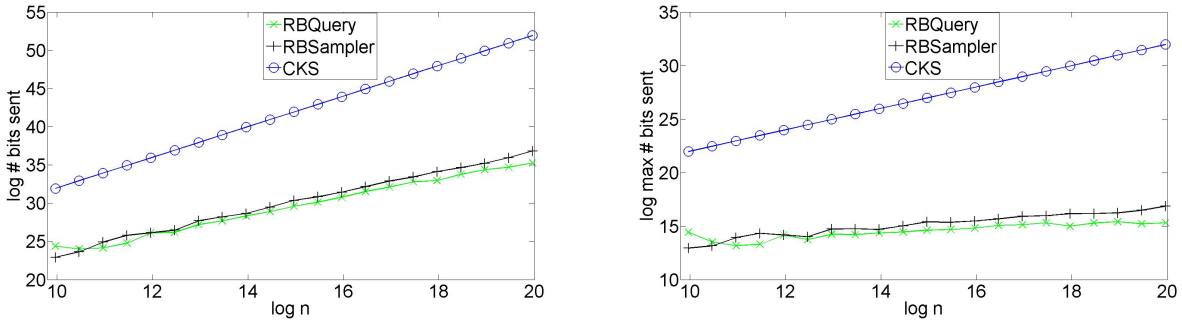

Fig. 2. Left: Log of number of nodes vs. number of bits sent; Right: Log of number of nodes vs max bits sent by a node.

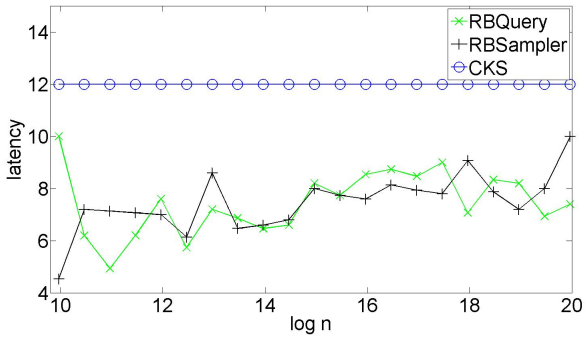

Fig. 3. Algorithm Latency

4.1 Experimental setup

We used the OpenMP compiler directives in conjunction with the C++11 multi-threading primitives to speed up the simulation on a machine with 128G of memory and 48 cores. Because of the computational resources used by the machine we tracked only the total messages sent and the maximum number of messages sent by any processor. The size of the network simulated was between 1,000 and 1,024,000 processors.

For our simulations of RBQUERY, we selected a value of the constants as dictated by our analysis in Section 3.1: $\epsilon_0 = 1/8$, $\epsilon > 1/6$, $C = 40$. We also made the size of the neighbors of a node in the algorithm to be $C \ln^2 n$. Increasing by a factor of $\ln n$ allows us to use a smaller C. With these parameter settings, we never observed RBQUERY fail to bring all processors to agreement in the experiments we ran. We simulated an adversary that causes each bad processor to send the same number of messages as a good processor

(i.e. $C \ln^2 n$); to send these messages to processors selected uniformly at random with replacement; and for each of these messages to contain a vote that is 0 with probability $1/2$ and 1 with probability $1/2$.

For our simulations of RBSAMPLER, we selected a value of the constants as dictated by our analysis in Section 3.2: $\epsilon_0 = 1/8$, $\epsilon > 1/6, C = 6, d = 3$, which implies that the size of the neighbors of a node in the algorithm is $C \ln^3 n$. With these parameter settings, we never observed RBSAMPLER fail to bring all processors to agreement in the experiments we ran. We simulated an adversary that causes each bad processor to send messages to all neighboring processors in the sampler; and for each of these messages to contain a vote that is 0 with probability $1/2$ and 1 with probability $1/2$.

We compare our algorithms with the CKS algorithm from the paper [4] which has $O(1)$ latency but requires all-to-all communication. Each data point in our points is an average over 30 trials.

We make the assumption that the latencies of our algorithms are dominated by the time it takes to send a message and we can use this as the unit cost of an operation. Bhatele and Kale [3] provide information via benchmarks about what these numbers might mean in real world applications. In [3], the values for message latencies are between 5 and 16 milliseconds for the largest message size in the paper (1MB).

4.2 Results

Figure 1 (left) shows the logarithm of the total messages sent for all algorithms simulated. We can see that RBQUERY and RBSAMPLER send less messages than CKS when the network size is larger than 5,000 processors. In particular, RBQUERY sends less messages than the other algorithms for network sizes greater than 2,000 processors. Note that RBQUERY sends slightly fewer messages than RBSAMPLER. The slope of the RBQUERY and RBSAMPLER graph is about 3.5 while that of the CKS is about 4.2.

Figure 1 (right) shows the maximum number of messages sent by a processor. This gives an idea of the load-balancing properties of the algorithm. The crossover point at which RBQUERY and RBSAMPLER have a smaller maximum number of messages sent than CKS is for a network size of 4,000 processors. RBQUERY is slightly better load-balanced than RBSAMPLER. The slope of the CKS graph is about 1.0, while that of the RBQUERY and RBSAMPLER algorithms is about 0.1.

Figure 2 (left) shows the total bits sent. For this criteria, RBQUERY and RBSAMPLER always perform well. The number of bits sent is dramatically less than that of CKS since the size of a message in CKS is about the same size as an RSA signature, which can be quite large. Again RBQUERY has a slight edge over RBSAMPLER. The slope of the graph for CKS is about 2.2, while RBQUERY and RBSAMPLER have a slope of about 1.3.

Figure 2 (right) shows the maximum bits sent by a processor. RBQUERY and RBSAMPLER again perform better than CKS for all the network sizes we consider. RBQUERY has a smaller maximum number of bits than RBSAMPLER. The slope of the graph for the CKS algorithm is about 1.0, while the slope for the RBQUERY and RBSAMPLER algorithms is about 0.4.

Figure 3 shows the latency of the algorithms. The latency of RBQUERY and RBSAMPLER are comparable and never exceed 10. RBQUERY and RBSAMPLER seem to have slightly smaller latency than CKS. The latencies for both the RBQUERY and RBSAMPLER vary since the latencies are random variables.

5 Conclusion

We have described two algorithms that make use of a random beacon to significantly reduce the bandwidth requirements needed to solve the Byzantine agreement problem. We have proven these algorithms correct and have demonstrated via simulations that, for large networks sizes, they are significantly more efficient than a state of the art algorithm. We believe our approach is likely to be useful for solving Byzantine agreement in large peer-to-peer networks.

Our results bring up several interesting questions for future research including the following.

- Can we show lower bounds on the total number of messages that must be sent to solve Byzantine agreement with a random beacon? Can we do better when the adversary is non-adaptive?
- What is the effect if the random beacon is not completely random? How much randomness is really required to compute Byzantine agreement sending $\tilde{O}(1)$ messages per processor?
- Our algorithms work in the partially synchronous model of communication. Is it possible to modify them so that they work in the completely asynchronous model?

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 Annual Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.
2. Yonatan Aumann and Michael O. Rabin. Information theoretically secure communication in the limited storage space model. In *In Advances in Cryptology, Proceedings of the 19th Annual International Cryptology Conference (CRYPTO*, pages 65–79, 1999.
3. Abhinav Bhatele and V. Laxmikant. An evaluative study on the effect of contention on message latencies in large supercomputers. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.
4. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132, New York, NY, USA, 2000. ACM.
5. Christian Cachin and Ueli Maurer. Unconditional security against memory-bounded adversaries. In *In Advances in Cryptology CRYPTO 97, Lecture Notes in Computer Science*, pages 292–306. Springer-Verlag, 1997.
6. Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. In *Proceedings of the 2010 international conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

7. Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985.
8. Stefan Dziembowski and Ueli Maurer. Tight security proofs for the bounded-storage model. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 341–350, New York, NY, USA, 2002. ACM.
9. D. Eastlake 3rd. Publicly Verifiable Nomcom Random Selection. RFC 2777 (Informational), February 2000. Obsoleted by RFC 3797.
10. D. Eastlake 3rd. Publicly Verifiable Nominations Committee (NomCom) Random Selection. RFC 3797 (Informational), June 2004.
11. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
12. Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In *Proceedings of DISC 2009:23rd International Symposium on Distributed Computing. Elche/Elx, Spain, September 23-25, 2009*, 2009.
13. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In *PODC*, pages 420–429. ACM, 2010.
14. Valerie King and Jared Saia. Scalable byzantine computation. *SIGACT News*, 41:89–104, September 2010.
15. John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.
16. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
17. Hui Huang Lee, Ee chien Chang, and Mun Choon Chan. Pervasive random beacon in the internet for covert coordination. In *In: Information Hiding, 7th International Workshop, IH 2005 (LNCS 3727*, pages 53–61. Springer, 2005.
18. Anna Lysyanskaya. Efficient threshold and proactive cryptography secure against the adaptive adversary (extended abstract).
19. Ueli M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5:53–66, 1992.
20. M. Rabin. Randomized Byzantine generals. In *Proc. Symposium on Foundations of Computer Science*, pages 403–409, 1983.
21. Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27:256–267, 1983.
22. Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: the OceanStore Prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, 2003.
23. Bitcoin Website. http://www.bitcoin.org/.
24. Bittorrent Website. http://www.bittorrent.com/.
25. Skype Website. http://www.skype.com/.