Load balanced Scalable Byzantine Agreement through Quorum Building, with Full Information

Valerie King[1], Steven Lonargan[1], Jared Saia[2], and Amitabh Trehan[1]

[1] Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, BC, Canada V8W 3P6,
`val@uvic.ca, {sdlonergan, amitabh.trehaan}@gmail.com`,

[2] Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386,
`saia@cs.unm.edu`

This research was partially supported by NSF CAREER Award 0644058, NSF CCR-0313160, AFOSR MURI grant FA9550-07-1-0532, and NSERC.

Abstract. We address the problem of designing distributed algorithms for large scale networks that are robust to Byzantine faults. We consider a message passing, full information model: the adversary is malicious, controls a constant fraction of processors, and can view all messages in a round before sending out its own messages for that round. Furthermore, each bad processor may send an unlimited number of messages. The only constraint on the adversary is that it must choose its corrupt processors at the start, without knowledge of the processors' private random bits.

A *good quorum* is a set of $O(\log n)$ processors, which contains a majority of good processors. In this paper, we give a synchronous algorithm which uses polylogarithmic time and $\tilde{O}(\sqrt{n})$ bits of communication per processor to bring all processors to agreement on a collection of n *good quorums*, solving Byzantine agreement as well. The collection is balanced in that no processor is in more than $O(\log n)$ quorums. This yields the first solution to Byzantine agreement which is both scalable and load-balanced in the full information model.

The technique which involves going from situation where slightly more than 1/2 fraction of processors are good and and agree on a short string with a constant fraction of random bits to a situation where all good processors agree on n good quorums can be done in a fully asynchronous model as well, providing an approach for extending the Byzantine agreement result to this model.

As an additional result, we present a fully asynchronous algorithm that can go from a situation where slightly more than 1/2 fraction of processors are good and and agree on a short string with a constant fraction of random bits, to a situation where all good processors agree on n good quorums, and can do this in a load balanced way. This provides an approach for extending the Byzantine agreement result above to the asynchronous model, with an additional $\log n$ factor in the number of bits communicated.

1 Introduction

The last fifteen years have seen computer scientists slowly come to terms with the following alarming fact: not all users of the Internet can be trusted. While this fact is hardly surprising, it is alarming. If the size of the Internet is unprecedented in the history of engineered systems, then how can we hope to address the challenging problem of scalability *and also* the challenging problem of resistance to malicious users?

Recent work attempts to address both of these problems concurrently. In the last few years, almost everywhere Byzantine agreement, i.e., coordination between all but a $o(1)$ fraction of processors, was shown to be possible with no more than polylogarithmic bits of communication per processor and polylogarithmic time [13]. More recently, scalable everywhere agreement was shown to be possible if a small set of processors took on the brunt of each communicating $\Omega(n^{3/2})$ bits to the remaining processors [11], or if private channels are assumed [12].

In this paper, we give the first load-balanced, scalable method for agreeing on a bit in the synchronous, full information model. In particular, our algorithm requires each processor to send only $\tilde{O}(\sqrt{n})$ bits. Our technique also yields an agreement on a collection of n *good quorum gateways* (referred to as quorums from now on), that is, sets of processors of size $O(\log n)$ each of which contains a majority of good processors, and a 1-1 mapping of processors to quorums. The collection is balanced in that no processor is in more than $O(\log n)$ quorums. Our usage of the quorum terminology is similar to that in the peer-to-peer literature [17,6,1,3,5,8], where quorums are of $O(\log n)$ size each having a majority of good processors, and allow for containment of adversarial behavior via majority filtering. Quorums are useful in an environment with malicious processors as they can act as a gateway to filter messages from by bad processors. For example, a bad processor x can be limited in the number of messages it sends if other processors only accept messages sent by a majority of processors in x's quorum, and the majority only agree to forward a limited number of messages from x.

The number of bits of communication required per processor is polylogarithmic to bring all but $o(1)$ processors to agreement and $\tilde{O}(\sqrt{n})$ per processor for everywhere agreement on the composition of the n quorums. Our result is with an adversary that controls up to a $1/3 - \epsilon$ fraction of processors, for any fixed $\epsilon > 0$, and which has full information, i.e., it knows the content of all messages passed between good processors. However, the adversary is non-adaptive, that is, it cannot choose dynamically which processors to corrupt based on its observations of the protocol's execution. Bad processors are allowed to send an unlimited number of bits and messages, and defense against a denial of service attack is one of the features of our protocol.

As an additional result, we present an asynchronous algorithm that can go from a situation where, for any positive constant γ, $1/2+\gamma$ fraction of processors are good and agree on a single string of length $O(\log n)$ with a constant fraction of random bits to a situation where all good processors agree on n good quorums. This algorithm is load-balanced in that it requires each processor to send only

$\tilde{O}(\sqrt{n})$ bits, and the resulting quorums are balanced in that no processor is in more than $O(\log n)$ quorums.

1.1 Methodology

Our synchronous protocol builds on a previous protocol which brings all but $o(1)$ processors to agreement on a set of $s = O(\log n)$ processors of which no more than a $1/3 - \epsilon$ fraction are bad, using a sparse overlay network [14]. Being few in number, these processors can run a heavyweight protocol requiring all-to-all communication to also agree on a string *globalstr* which contains a bit (or multiple bits) from each processor, such that a $2/3 + \epsilon$ fraction of the bits are randomly set. This string can be communicated scalably to almost all processors using a communication tree formed as a byproduct of the protocol (See [13,14]).

When a clear majority of good processors agree on a value, a processor should be able to learn that value, with high probability, by polling $O(\log n)$ processors. However the bad processors can thwart this approach by flooding all processors with requests. Even if there are few bad processors, in the full information model, the bad processors can target processors on specific good processors' poll lists to isolate these processors. To address this problem, we use the *globalstr* to build quorums to limit the number of effective requests. We also restrict the design of poll lists, preserving enough randomness that they are reliable, but limit the adversary's ability to target.

Key to our work here is that we show the existence of an averaging sampler type function, H, which is known at the start by all the processors and which with high probability, when given an $O(\log n)$ length string with a constant fraction of random bits, and a processor ID, produces a good quorum for every ID. Our protocol then uses the fact that almost all processors agree on a collection of good quorums to bring all processors to agree on the string in a load balanced manner, and hence the collection of quorums. Similarly, to solve Byzantine agreement, a single bit agreed to by the initial small set can be agreed to by all the processors. We also show the existence of a function J which uses individually generated random strings and a processor's ID to output a $O(\log n)$ poll list, so that the distribution of poll lists has desired properties.

These techniques can be extended to the asynchronous model assuming a scalable implementation of [10]. That work shows that a set of size $O(\log \log n)$ processors with $2/3 + \epsilon$ good processors can be agreed to almost everywhere with probability $1 - o(1)$. Bringing these processors to agreement on a string with some random bits is trickier in the asynchronous full information model, where the adversary can prevent a fraction of the good processors from being heard based on their random bits. However, [10] shows that it is possible to bring such a set to agreement on a string with some randomness, which we show is enough to provide a good input to H.

1.2 Related work

Several papers are mentioned above with related results. Most closely related is the algorithm in [11] which similarly starts with almost everywhere agreement on a bit and a small representative set of processors from [13,14] and produces everywhere agreement. However it is not load balanced, and does not create quorums or require the use of specially designed functions H and J. With private channels, load balancing in the presence of an adaptive adversary is achievable with $\tilde{O}(\sqrt{n})$ bits of communication per processor [12].

Awerbuch and Scheidler have done important work in the area of maintaining quorums [3,4,5,6]. They show how to scalably support a distributed hash table (DHT) using quorums of size $O(\log n)$, where processors are joining and leaving, a functionality our method does not support. The adversary they consider is nonadaptive in the sense that processors cannot spontaneously be corrupted; the adversary can only decide to cause a good processor to drop out and decide if an entering processor is good or bad. A critical difference between their results and ours is that while they can maintain a system that starts in a good configuration, they cannot initialize such a system unless the starting processors are all good. This is because an entering processor must start by contacting a good processor in a good quorum. The quorum uses secret sharing to produce a random number to assign or reassign new positions in a sparse overlay network (using the cuckoo rule [15]). These numbers and positions are created using a method for secret sharing involving private channels and cryptographic hardness assumptions.

In older work, Upfal, Dwork, Peleg and Pippenger addressed the problem of solving almost-everywhere agreement on a bounded degree network [16,7]. However, the algorithms described in these papers are not scalable. In particular, both algorithms require each processor to send at least a linear number of bits (and sometimes an exponential number).

1.3 Model

We assume a fully connected network of n processors, whose IDs are common knowledge. Each processor has a private coin. Communication channels are authenticated, in the sense that whenever a processor sends a message directly to another, the identity of the sender is known to the recipient, but we otherwise make no cryptographic assumptions. We assume a *nonadaptive* (sometimes called *static*) adversary. That is, the adversary chooses the set of tn bad processors at the start of the protocol, where t is a constant fraction, namely, $1/3 - \epsilon$ for any positive constant ϵ. The adversary is *malicious*: bad processors can engage in any kind of deviations from the protocol, including false messages and collusion, or crash failures, and bad processors can send *any* number of messages. Moreover, the adversary chooses the input bits of every processor. The good processors are those that follow the protocol.

We consider both synchronous and asynchronous models of communication. In the *synchronous* model, communication proceeds in rounds; messages are all sent out at the same time at the start of the round, and then received at the same

time at the end of the same round; all processors have synchronized clocks. The time complexity is given by the number of rounds. In the *asynchronous* model, each communication can take an arbitrary and unknown amount of time, and there is no assumption of a joint clock as in the synchronous model. The adversary can determine the delay of each message and the order in which they are received. We follow [2] in defining the running time of an asynchronous protocol as the time of execution, where the maximum delay of a message between the time it is sent and the time it is processed is assumed to be one unit.

We assume *full information*: in the synchronous model, the adversary is *rushing*, that is, it can view all messages sent by the good processors in a round before the bad processors send their messages in the same round. In the case of the asynchronous model, the adversary can view any sent message before its delay is determined.

1.4 Results

We use the phrase *with high probability* to mean that an event happens with probability at least $1 - 1/n^c$, for any constant c and sufficiently large n.

We show:

Theorem 1 (Synchronous Byzantine Agreement). *Let n be the number of processors in a synchronous full information message passing model with a nonadaptive, rushing adversary that controls less than $1/3 - \epsilon$ fraction of processors. For any positive constant ϵ, there exists a protocol which w.h.p. computes Byzantine agreement, runs in polylogarithmic time, and uses $\tilde{O}(\sqrt{n})$ bits of communication per processor.*

This result follows from the application of the load balanced protocol in [14], followed by the synchronous protocol introduced in Section 3 of this paper.

Theorem 2 (Almost everywhere to everywhere–asynchronous). *Let n be the number of processors in a fully asynchronous full information message passing model with a nonadaptive adversary. Assume that $(1/2 + \gamma)n$ good processors agree on a string of length $O(\log n)$ which has a constant fraction of random bits, and where the remaining bits are fixed by a malicious adversary after seeing the random bits. Then for any positive constant γ, there exists a protocol which w.h.p. brings all good processors to agreement on n good quorums; runs in polylogarithmic time; and uses $\tilde{O}(\sqrt{n})$ bits of communication per processor. Furthermore, if we assume that same set of good processors have agreed on an input bit (to the Byzantine agreement problem) then this same protocol can bring all good processors to agreement on that bit.*

A scalable implementation of the protocol in [10] following the lines of [14] would create the conditions in the assumptions of this theorem with probability $1 - O(1/\log n)$ in polylogarithmic time and bits per processor with an adversary that controls less than $1/3 - \epsilon$ fraction of processors. Then this theorem would yield an algorithm to solve asynchronous Byzantine agreement with probability $1 - O(1/\log n)$. The protocol is introduced in Section 4 of this paper.

2 Combinatorial Lemmas

Before presenting our protocol, we discuss here the properties of some combinatorial objects we shall use in our protocol.

Let $[r]$ denote the set of integers $\{1, \ldots, r\}$, and $[s]^d$ the multisets of size d consisting of elements of $[s]$. Let $H : [r] \to [s]^d$ be a function assigning multisets of size d to integers. We define the intersection of a multiset A and a set B to be the number of elements of A which are in B. H is a (θ, δ) *sampler* if at most a δ fraction of all inputs x have $\frac{|H(x) \cap S|}{d} > \frac{|S|}{s} + \theta$.

Let $r = n^{c+1}$. Let $i \in [n^c]$ and $j \in [n]$. Then we define $H(i,j)$ to be $H(in+j)$ and $H(i,*)$ to be the collection of subsets $H(i+1), H(i+2), ..., H(i+n)$.

Lemma 1 ([[9], Lemma 4.7], [[18], Proposition 2.20]). *For every $s, \theta, \delta > 0$ and $r \geq s/\delta$, there is a (θ, δ) sampler $H : [r] \to [s]^d$ with $d = O(\log(1/\delta)/\theta^2)$.*

A corollary of the proof of this lemma shows that if one increases the constant in the expression of d by a factor of c, we get the following:

Corollary 1 *Let $H[r]$ be constructed by randomly selecting with replacement d elements of $[s]$. For every $s, \theta, \delta, c > 0$ and $r \geq s/\delta$, for $d = O(\log(1/\delta)/\theta^2)$, $H(r)$ is a (θ, δ) sampler $H : [r] \to [s]^d$ with probability $1 - 1/n^c$.*

Lemma 2. *Let $r = n^{c+1}$ and $s = n$. Let $H : [r] \to [s]^d$ be constructed by randomly selecting with replacement d elements of $[s]$. Call an element $y \in [s]$ overloaded by H if its inverse image under H contains more than a.d elements, for some fixed element $a \geq 6$. The probability that any $y \in [s]$ is overloaded by any $H(i,*)$ is less than $1/2$, for $d = O(\log n)$ and $a = O(1)$.*

Proof. Fix i. The probability that the size of the inverse image of $y \in [s] \in H(i,*)$ is a times its expected size of d is less than 2^{-ad}, for $a \geq 6$, by a standard Chernoff bound. The probability that for any i that any $y \in [s]$ is overloaded is less than $n(n^c)2^{-ad} < 1/2$, by a union bound over all $y \in [s]$ and all i for $d = O(\log n)$.

Let S be any subset of $[n]$. A *quorum* or *poll list* is a subset of $[n]$ of size $O(\log n)$ and a *good quorum (resp., poll list) with respect to $S \in [n]$* is a quorum (resp., poll list) with greater than $1/2$ elements in S. Taking the union bound over the probabilities of the events given in the preceding corollary and lemma, and applying the probabilistic method yields the existence of a mapping with the desired properties:

Lemma 3. *For any constant c, there exists a mapping $H : [n^{c+1}] \to [n]^d$ such that for every i the inverse image of every element under $H[i,*]$ is $O(\log n)$ and for any choice of any subset $S \subset n$ of size at least $1/2 + \epsilon n$, with probability $1 - 1/n^c$ over the choice of random numbers $i \in [n^c]$, $H[i,*]$ contains all good quorums.*

The following lemma is needed to show results about the polling lists, which are subsets of size $O(\log n)$ just like quorums, but are used for a different purpose in the protocol.

Lemma 4. *There exists a mapping* $J : [1..., n^{c+1}] \rightarrow [n]^d$ *such that for any set of* $1/2 + \epsilon$ *fraction of good processors in* $[1 \dots n]$*:*

1. *At least* $n^{c+1} - n$ *elements of* L *are mapped to a good PollList.*
2. *For any* $L' \subset [n^{c+1}]$, $|L'| \leq n$, *let* R' *be any subset of* $[r]$, $|R'| \leq \epsilon|L'|/e^2$ *and let* L' *be the inverse image of* R' *under* J*. Then* $\sum_{x \in L'} |J(x) \cap R'| < \epsilon d|L'|/2$*. Hence* $|L'|/2$ *pollLists contain fewer than* ϵd *elements in* R'*.*

Proof. Part 1: The probability that a randomly constructed J has this property with probability greater than 1/2 follows from Lemma 3.

Part 2: Let J be constructed randomly as in the previous proofs. Fix L', fix R'.

$Pr[\sum_{x \in L'} |J(x) \cap R'| \geq \epsilon d|L'|] = \binom{d|L'|}{\epsilon d|L'|}(|R'|/n)^{\epsilon d|L'|} \leq [(e/\epsilon)(|R'|/n)]^{\epsilon d|L'|} \leq e^{-\epsilon d|L'|}$, for $|R'| \leq \epsilon n/e^2$

The number of ways of choosing a subset of size x and y from $[n^c]$ and $[n]$, resp., is bounded above by $(en^c/x)^x * (en/y)^y = e^{x(c \log n - \log x + 1) + y(\log n - \log y + 1)} < e^{2|L'|c \log n}$.

The union bound over all sizes of $x \leq n$ and y is less than 1/2 for $d > (2c \log n)/\epsilon + 1/\epsilon|L|$

Hence with probability less than 1/2, $\sum_{x \in L'} |J(x) \cap R'| > \epsilon d|L'|$ for all subsets L' of size n or less in $[n^c]$ and all subsets R' of size $\epsilon|L'|/e^2$.

Finally, by the union bound, a random J has both properites (1) and (2) with probability greater than 0. By the probabilistic method, there exists a function J with properties (1) and (2).

2.1 Using the Almost-everywhere agreement protocol in [13,14]

We observe that this protocol which uses polylogarithmic bits of communication generates a representative set S of $O(\log n)$ processors which is agreed upon by all but $O(1/\log n)$ fraction of good processors, and any message agreed upon by the processors is learned by all but $O(1/\log n)$ fraction of good processors. Hence we start in our current work from the point where there is an $b \log n$ bit string *globalstr* agreed upon by all but $O(1/\log n)$ fraction of good processors such that $2/3+\epsilon$ fraction of good processor in S have each generated c'/b random bits (see below), and the remaining bits are generated by bad processors after seeing the bits of good processors. The ordering of the bits is independent of their value and is given by processor ID. *globalstr* is random enough:

Lemma 5. *With probability at least* $1 - 1/n^c$*, for sufficiently large constant* c' *and* $d = O \log n)$*, there is an* $H : [n^{c'+1}] \rightarrow [n]^d$ *such that* $H(globalstr, *)$ *is a collection of all good quorums.*

Proof. By Lemma 3 there are n^c good choices for *globalstr* and n bad choices. We choose c' to be a multiple of b which is greater than $(3/2)c$. Fix one bad choice string. The probability of the random bits matching this string is less than $2^{-(2/3c' \log n)}$ and by a union bound, the probability of it matching any of the n bad strings is less than $1/n^c$.

3 Algorithm

In this section, we describe the protocol (Protocol 3.1) that reaches everywhere agreement from almost everywhere agreement.

Given: Functions H (as described in Lemma 3), and J (as described in Lemma 4).

Part I: Setting up Candidate Lists.

1: **for** each processor p: **do**
2: Select uniformly at random a subset, $samplelist_p$, of processor IDs where $|samplelist_p| = c\sqrt{n}\log(n)$.
3: $p.send(samplelist_p, < candstr_p >)$.
4: Set $candlist_p \leftarrow candstr_p$.
5: For each processor r that sent $< candstr_r >$ to p, add $candstr_r$ to $candlist_p$ with probability $1/\sqrt{n}$.

Part II: Setting up Requests through quorums.

1: **for** each processor p: **do**
2: p generates a random string $rstr_p$.
3: For each candidate string $s \in candlist_p$, $p.send(\mathrm{H}(s,p), < rstr_p >)$.
4: Let $polllist_p \leftarrow \mathrm{J}(rstr_p, p)$
5: **if** processor $z \in \mathrm{H}(candstr_z, p)$) and $z.accept(p, < rstr_p >)$ **then**
6: **for** each processor $y \in polllist_p$ **do**
7: $z.send(\mathrm{H}(candstr_z, y), < p \rightarrow y >)$
8: **for** Processor $t \in \mathrm{H}(candstr_t, y)$ for any processor y **do**
9: $\mathrm{Requests}_t(y) = \{< p \rightarrow y > \ |$ received from p 's quorum $\mathrm{H}(candstr_t, p)\}$

Part III: Propagating ***globalstr*** **to every processor.**

1: **for** $\log n$ rounds in parallel **do**
2: **if** $0 < |\mathrm{Requests}_t(y)| < c' log(n)$ **then**
3: **for** $< p \rightarrow y > \in \mathrm{Requests}_t(y)$ **do**
4: $t.send(y, < p \rightarrow y >)$
5: set $\mathrm{Requests}_t(y) \leftarrow \emptyset$.
6: **if** $y.accept(\mathrm{H}(candstr_y, y), < p \rightarrow y >)$ **then**
7: $y.send(p, < candstr_y >)$
8: $y.send(\mathrm{H}(candstr_y, p), < candstr_y >)$
9: **when** for processor p, count of processors in $polllist_p$ sending candidate string s over all rounds reaches a majority: Set $candstr_p \leftarrow s$.
10: **if when** for processor $z \in \mathrm{H}(candstr_z, p)$, count of processors in $polllist_p$ sending string s over all rounds reaches a majority **then**
11: **for** Processor $y \in polllist_p$ such that y did not yet respond **do**
12: $z.send(\mathrm{H}(candstr_y, z), < Abort, p >)$
13: **if** $t \in \mathrm{H}(candstr_t, y)$ and $t.accept(\mathrm{H}(candstr_t, p), < Abort, p >$ **then**
14: $< p \rightarrow y >$ is removed from $\mathrm{Requests}_t(y)$.

Protocol 3.1: Load balanced almost everywhere to everywhere

3.1 Description of the algorithm

Precondition: Each processor p starts with a hypothesis of the global string, $candstr_p$; this hypothesis may or may not equal *globalstr*. However, we make a critical assumption that at least $1/2+\gamma$ fraction of processors are good and knowledgable i.e. their *candstr* equals *globalstr*. Actually we can ensure that $2/3+\epsilon-O(1/\log n)$ fraction of processors are good and knowledgeable using the almost-everywhere protocol from [13,14], but we need only have $1/2+\epsilon$ fraction for our protocol to work.

Let $candlist_p$ be a list of candidate strings that p collects during the algorithm. Further, we call $\mathrm{H}(candstr_q, p)$ a *quorum* of p (or p's quorum) according to q. If a processor p is sending to a quorum for x then it is assumed to mean that this is the quorum according to p, unless otherwise stated. Similarly, if t is sending conditional on its being in a particular quorum, then we mean this quorum according to t. Often, we shall denote a message within arrow brackets ($<>$), in particular $< p \rightarrow y >$ is the message that p has requested information from y. We call a quorum a *legitimate quorum* of p if it is generated by the *globalstr* i.e. $\mathrm{H}(globalstr, p)$.

We also define the following primitives:

- $v.send(X, m)$: Processor v sends message m to all processors in set X.
- $v.accept(X, m)$: Processor v accepts the message m received from a majority of the processors in the set X (which could be a singleton set), otherwise It rejects it.
- *Rejection of excess*: Every processor will reject messages received in excess of the number of those messages dictated by the protocol in that round or stage of execution of the protocol.

We assume each processor knows H and J. The key to achieving reliable communication channels through quorums is to use the *globalstr*. To begin, each processor p sends its candidate string $candstr_p$ directly to $c\sqrt{n}\log n$ randomly selected processors (the $samplelist_p$). It then generates its own list of candidates $candlist_p$ for the *globalstr* including $candstr_p$ and every received string with probability $1/\sqrt{n}$. This ensures that p has at least one *globalstr* in its list.

The key to everywhere agreement is to be able to poll enough processors reliably so as to be able to learn *globalstr*. Part II sets up these polling requests. Each processor p generates a random string $rstr_p$, which is used to generate p's poll list $polllist_p$ using the function J by both p and its quorums. All the processors in the poll list are then contacted by p for their candidate string. In line 2, p determines its quorum for each of the strings in its $candlist_p$ and sends $rstr_p$ to the processors in the quorums. To prevent the adversary from targeting groups of processors, the quorums do not accept the poll list but rather the random string and then generate the poll list themselves. The important thing to note here is that even if p sent a message to its quorum the processors in the quorum will not accept the messages unless according to their own candidate string, they are in p's quorum. Hence, it is important to note that w.h.p. at least one of these quorums is a legitimate quorum.

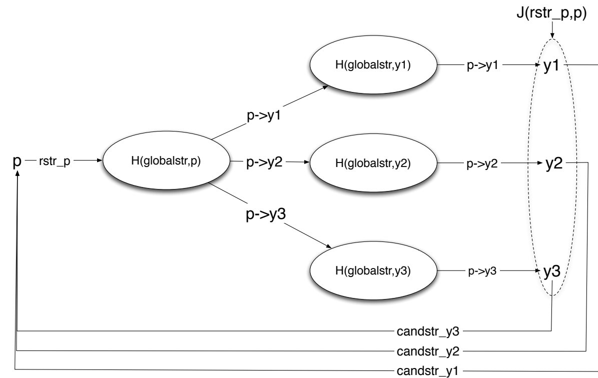

Fig. 1. Example run of Parts II and III of Algorithm 3.1

Since p sends to at least one legitimate quorum, and the processors in this quorum will accept p's request, this request will be forwarded. p's quorum in turn contacts processor y's quorum for each y that was in p's poll list. The processors in y's legitimate quorum gather all the requests meant for y in preparation for the next part of the protocol.

Part III proceeds in $\log n$ rounds. The processors in y's quorum only forward the received requests if they number less than $c' \log n$ for some fixed constant c'. This prevents any processor from being overloaded. Once y accepts the requests (in accordance with $y.accept$), it will send its candidate string directly to p and also to p' s quorum. When p gets the same string from a majority of processors in its poll list, it sets its own candidate string to this string. This new string w.h.p. is $globalstr$. There may be overloaded processors which have not yet answered p's requests. To release the congestion, p will send *abort* messages to these quorums, which will now take the request off p's request off their list. In each round, the number of satisfied processors falls by at least half, so that no more than $\log n$ rounds are needed. In this way, w.h.p. each processor decides the $globalstr$.

Figure 1 illustrates an example run of parts II and III of Algorithm 3.1. For clarity, in this figure, all processors shown are assumed to be correct; only messages sent to legitimate quorums are shown; and a single arrow to a quorum illustrates a message sent to all processors in the quorum. The illustration begins with processor p sending the message $rstr_p$ to the quorum $H(globalstr, p)$. The messages $rstr_p$ that are sent to H(s, p), for other $s \in candlist_p; s \neq globalstr$ are not shown. In this example, $J(rstr_p, p) = \{y1, y2, y3\}$, i.e. a set consisting of just 3 processors. In the next step of the algorithm, processors in $H(globalstr, p)$ send the messages $< p \rightarrow y1 >$, $< p \rightarrow y2 >$, $< p \rightarrow y3 >$ to the quorums $H(globalstr, y1)$, $H(globalstr, y2)$, $H(globalstr, y3)$ respectively. Next these quorums forward the messages to the appropriate processors in $J(rstr_p, p)$. Finally, the processors in $J(rstr_p, p)$ send their *candstr* values directly to p.

3.2 Proof of correctness

The conditions for the correctness of the protocol given in Protocol 3.1 are stated as Lemma 10. To prove that, first we show the following supporting lemmas.

Lemma 6. *W.h.p., at least one string in the $candlist_p$ of processor p is the globalstr.*

Proof. The proof of this follows from the birthday paradox. If there are n possible birthdays and $O(\sqrt{n})$ children, two will likely share a birthday. Adding an $O(\log n)$ factor increases the probability for this to happen n times w.h.p.

Lemma 7. *For processor p and its random string $rstr_p$, a majority of the processors y in $polllist_p$ are good and knowledgable, and they receive the request $< p \rightarrow y >$.*

Proof. The poll list for processor p, $polllist_p$ is generated by the sampler J using p's random string $rstr_p$ and p's ID. By Lemma 4, a majority of $polllist_p$ is good and knowledgable.

From Lemmas 5 and 6, processor p will send its message for its poll lists to at least one legitimate quorum. Since a majority of these are good and knowledgable, they will forward the message $< p \rightarrow y >$ for each processor $y \in polllist_p = \mathrm{J}(rstr_p, p)$ to at least one legitimate quorum of y. By Lemma 9, y shall accept the message.

Observation 1 *The messages sent by the bad processors, or good but not knowledgable processors (having candstr $\neq$ globalstr) do not affect the outcome of the protocol.*

Proof. All communication in Parts 2 and 3 is verified by a processor against its quorums or poll list. Any communication received through the quorum or poll list is influential if only a majority of processors in them have sent it (either using the *accept* primitive or by counting total messages received). By Lemmas 6 and 7, majority of these lists are good and knowledgable.

Lemma 8. *For the protocol, any processor sends no more than $\tilde{O}(\sqrt{n})$ bits.*

Proof. Consider a good and knowledgeable processor p. In Part-I, line 3, p sends $c\sqrt{n}\log n$ messages. For part II of the algorithm, consider p is in the quorum of a processor z; p forwards $O(\log^2 n)$ messages to the quorums of z's poll list. In part III, p forwards only $O(\log n)$ requests to z. The cost of aborting is no more than the cost of sending. In addition, z answers no more than the number of requests that its quorum forwards. By the rejection of excess primitive, no extra messages are sent. Thus, p sends at most $\tilde{O}(\sqrt{n})$ bits over a run of the whole protocol.

Lemma 9. *By the end of part III, for each p, a majority of p's poll list have received p's request to respond from their legitimate quorums.*

Proof. Quorums will forward requests provided their processors are not overloaded. We show by induction that if in round i, there were x processors making requests to overloaded processors, there are no more than $x/2$ requests to overloaded processors in round $i+1$, and thus in $\log n$ rounds, there shall be no overloaded processors. Hence every processor will answer its requests. Refer to Lemma 4: let R_i be the set of overloaded processors in round i (those that have more than $(4/\epsilon)d$ requests). Consider the set L_i of processors which made these requests; $|L_i| \geq 8/\epsilon|R_i|$. By part 2 of the lemma, half the processors in L_i contain less than ϵ fraction of their PollLists in R_i, and their requests will be satisfied in the current round by a majority of good processors. Thus, there are now no more than $|L_i|/2$ such processors making requests to processors in R_i, and hence to overloaded processors in round $i+1$.

Lemma 10. *Let n be the number of processors in a synchronous full information message passing model with a nonadaptive rushing malicious adversary which controls less than a $\frac{1}{3}-\epsilon$ fraction of processors, and more than $\frac{1}{2}+\gamma$ fraction of processors are good and knowledgable. For any positive constants ϵ, γ there exists a protocol w.h.p. such that: 1) At the end of the protocol, each good processor is also knowledgable, 2) The protocol takes no more than $O(\log n)$ rounds in parallel, using no more than $\tilde{O}(\sqrt{n})$ messages per processor.*

Proof. Part 1 follows from Lemmas 7 and observation 1; processor p hears back from its poll list and becomes knowledgable. Part 2 follows directly from lemmas 9 (Protocol is completed in $O(\log n)$ rounds) and 8.

4 Asynchronous version

The asynchronous protocol for Byzantine agreement relies on the *globalstr* being generated by a scalable version of [10]. Such a string would have a reduced constant fraction of random bits but there would still be sufficient randomness to guarantee the properties needed. Note that the reduction in the fraction of random bits needed in the string can be compensated for by increasing the length of the string in the proof of Lemma 5.

The asynchronous protocol to bring all processors to agreement on the *globalstr* can be constructed from the synchronous protocol by using the primitive *asynchaccept* instead of *accept* and by changes to Part III. The modified Part III is shown as Protocol 4.1. The primitive $v.asynchaccept(X, m)$ is defined as : Processor v waits until $|X|/2+1$ messages which agree on m are received and then takes their value. In Part III, since there are no rounds, there is instead an end-of-round signal for each "round" which is determined when enough processors have decided. the quorums are organized in a tree structure which allows them to simulate the synchronous rounds by explicitly counting the number of processors that become knowledgable. Round number is determined by the count of quorums which have received $n/2+1$ answers to requests of their processor. The quorum of a processor monitors the number of requests received and only forward the requests to a processor when the current number of requests received

in a round is sufficiently small. The asynchronous protocol incurs an additional overhead of a $\log n$ factor in the number of messages.

The quorums are organized into a rooted binary tree T with their placement determined by their processor's ID*, with a "root quorum" at the root.*

Repeat each time an end-of-round signal is received while there are any non knowledgable processors. Each execution is referred to as a *round* . Let n be number of non knowledgable processors at beginning of a round:

1. Let $\mathrm{R}_t(y)$ = number of requests received in the current round by processor t in y's quorum, for y + overflow messages from previous round.
2. if $(0 < \mathrm{R}_t(y) < c' logn)$ then $t.send(y, < p \rightarrow y >)$
   else accumulate incoming accepted messages to an overflow list for next round.
3. if $y.asynchaccept(\mathrm{H}(candstr_y, y), < p \rightarrow y >)$, then $y.send(p, < candstr_y >)$ and $y.send(\mathrm{H}(candstr_y, p), < candstr_y >)$
4. if, for p, | processors in $polllist_p$ sending candidate string s | reaches $(1/2) polllist_p + 1$
   (a) Set $candstr_p \leftarrow s$.
   (b) for all {Processor $y \in polllist_p$ that did not yet respond}, $z.send(\mathrm{H}(candstr_y, z), < Abort, p >)$ where z are the processors in p's gateway.
   (c) if $t \in \mathrm{H}(candstr_t, y)$ and $t.accept(\mathrm{H}(candstr_t, p), < Abort, p >)$ then $< p \rightarrow y >$ is removed from $overflowlist_t(y)$.
   (d) p's quorum's sends a message (directly) to the root quorum.
5. When the root quorum hears from $n/2$ quorums, it sends an end-of-round signal down the binary tree to all the quorums.

Protocol 4.1: Asynchronous Part-III

4.1 Proof of correctness and termination

If all processors decide on a value for the *globalstr*, then w.h.p. they decide correctly. This follows from the proof for the synchronous model. The *globalstr* can be extended to include an extra 1 bit which is the Byzantine agreement value that was decided by the almost everywhere Byzantine agreement protocol. This attachment is disregarded until the end of the protocol when the processors use this value to decide on solution to the Byzantine agreement problem.

We now show that w.h.p. every processor decides. First note that Parts I and II of the synchronous protocol are inherently asynchronous. That is, as messages are received, they can be responded to in whatever order they arrive. A "round" ends in the asynchronous Part III if sufficient number of quorums receive answers to requests. Eventually all good processors which send messages will have their messages delivered, and the number of quorums will reach $n/2$. Finally the end-of-message signal will be successfully forwarded down to the leaves of the binary tree since there is a majority of good processors in each

quorum which will forward it. If we let the maximum time delay from sending a message to delivery of the message be one time step, then each round in Part III takes $O(\log n)$ time steps and polylogarithmic bits. The time is dominated by the need to forward the end-of-round signal down the tree of quorums, which is a binary tree of height $\log n$.

5 Future work

Handling churn is an important concern. We note that the probability of success of Awerbuch and Scheidler's work is ensured for only a limited polynomial number of joins and leaves. A natural interesting question to ask is whether our techniques could be combined with those in [6] to create and maintain quorums and a DHT indefinitely in the presence of churn, without relying on complexity assumptions or private channels.

References

1. James Aspnes and Gauri Shah. Skip graphs. In *SODA*, pages 384–393, 2003.
2. Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.
3. B. Awerbuch and C. Scheideler. Provably secure distributed name service. In *ICALP*, 2004.
4. B. Awerbuch and C. Scheideler. Robust distributed name service. In *IPTPS*, 2004.
5. B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. In *SPAA*, 2006.
6. B Awerbuch and C Scheideler. Towards a scalable and robust dht. *Theory Comput. Syst.*, 45(2):234–260, 2009.
7. C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. In *STOC*, page 379, 1986.
8. Amos Fiat, Jared Saia, and M. Young. Making Chord Robust to Byzantine Attacks. In *ESA*, 2005.
9. Ronen Gradwohl, Salil P. Vadhan, and David Zuckerman. Random selection with an adversarial majority. In *CRYPTO*, pages 409–426, 2006.
10. Bruce M. Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. In *SODA*, pages 1038–1047, 2008.
11. Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{o}(n^{3/2})$ bits. In *DISC*, pages 464–478, 2009.
12. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. In *PODC*, 2010.
13. Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.
14. Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *FOCS*, pages 87–98, 2006.
15. C. Scheideler. How to Spread Adversarial Nodes? Rotate! In *STOC*, 2005.
16. E. Upfal. Tolerating linear number of faults in networks of bounded degree. In *PODC*, pages 83–89, 1992.

17. M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical robust communication in DHTs tolerating a byzantine adversary. In *ICDCS*, pages 263–272. IEEE, 2010.
18. David Zuckerman. Randomness-optimal oblivious sampling. *Random Struct. Algorithms*, 11(4):345–367, 1997.