From Almost Everywhere to Everywhere: Byzantine Agreement with $\tilde{O}(n^{3/2})$ Bits

Valerie King[1,⋆] and Jared Saia[2,⋆⋆]

[1] Dept. of Computer Science, University of Victoria
P.O. Box 3055, Victoria, BC, Canada V8W 3P6
val@uvic.ca
[2] Department of Computer Science, University of New Mexico
Albuquerque, NM 87131-1386
saia@cs.unm.edu

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 synchronous 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 corrupt 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. To the authors' best knowledge, there have been no protocols for such a model that compute Byzantine agreement without all-to-all communication, even if private channels or cryptography are assumed, unless corrupt processors' messages are limited.

In this paper, we give a polylogarithmic time algorithm to agree on a small representative committee of processors using only $\tilde{O}(n^{3/2})$ total bits which succeeds with high probability. This representative set can then be used to efficiently solve Byzantine agreement, leader election, or other problems. This work extends the authors' work on scalable almost everywhere agreement.

1 Introduction

Increases in frequency, speed and severity of attacks on the Internet have led to a resurgence of interest in the Byzantine fault model for very large networks, see for example [3,14]. The goal of this work is to address the problem of designing distributed algorithms for large scale networks that are robust to Byzantine faults.

Our paper concerns the well-studied message-passing model: n processors are in a fully connected network and a malicious adversary with full information controls less than a $1/3 - \epsilon$ fraction of these processors, where ϵ is any positive constant. Our main contribution is to show that randomization can be used to

⋆ This research was supported by NSERC.
⋆⋆ This research was partially supported by NSF CAREER Award 0644058 and NSF CCR-0313160 Award.

I. Keidar (Ed.): DISC 2009, LNCS 5805, pp. 464–478, 2009.

break the 1985 $\Omega(n^2)$ barrier [4] for message and bit complexity for Byzantine agreement in the deterministic synchronous model, if we assume the adversary's choice of bad processors is made at the start of the protocol, i.e., independent of processors' private coinflips. Our techniques lead to solutions with $\tilde{O}(n^{3/2})$ bit complexity for leader election and universe reduction. Our protocols are polylogarithmic in time and, except for leader election, succeed with high probability.

We overcome the lower bound of [4] by allowing for a small probability of error. In particular, the lower $\Omega(n^2)$ lower bound on the number of messages to compute Byzantine agreement deterministically implies that any randomized protocol which computes Byzantine agreement with $o(n^2)$ messages must err with some probability $\rho > 0$, since with probability $\rho > 0$, an adversary can guess the random coinflips and cause the protocol to fail when those coinflips occur. Thus, any randomized algorithm achieving $o(n^2)$ messages must necessarily be a Monte Carlo algorithm.

In 2006, the authors [12] showed that almost everywhere Byzantine agreement, where $(1-1/\log n)$ fraction of the good processors come to agreement on a good processor's input bit, could be computed with high probability in polylogarithmic time with a polylogarithmic number of bits of communication per processor. It is easy to see that one round suffices to go from almost everywhere agreement to everywhere agreement with $n(n-1)$ additional bits of communication. Each processor sends every other processor its bit, and each processor decides on the majority. Is there a way to avoid this last high cost round?

The difficulty of achieving $o(n^2)$ messages is illustrated by showing what goes wrong with the obvious approach: each processor randomly selects $O(\log n)$ processors to poll and decides a value equal to the majority of their responses. The problem with this protocol is *flooding*. That is, bad processors may all bombard every processor for requests and no processor will be able to respond to all the requests without incurring a cost of $\Theta(n^2)$ messages. Previous to this paper we did not know of any technique of flood avoidance other than to design a protocol in which each processor predetermines (perhaps using private random bits) at the start of each round the list of processors it is willing to listen to. That is, this list does not depend on the list of processors who actually send. This paper uses a novel technique to deal with flooding that may be of independent interest.

1.1 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 t bad processors at the start of the protocol, where t is a constant fraction, namely, $1/3-\epsilon$ for any positive constant ϵ of the number of processors n. The adversary is *malicious*: it chooses the input bits of every processor, bad processors can engage in any kind of deviations from the protocol, including false messages and collusion, or crash

failures, while the remaining processors are good and follow the protocol. Bad processors can send *any* number of messages.

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 [1] in defining the running time of an asynchronous protocol as the time of execution, assuming 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 asynchronous model, the adversary can view any sent message before its delay is determined.

1.2 Problems

One of the most well studied problems in distributed computing is the *Byzantine agreement* problem. In this problem, each processor begins with either a 0 or 1. An execution of a protocol is *successful* if all processors terminate and, upon termination, agree on a bit held by at least one good processor at the start. The *leader election* problem is the problem of all processors agreeing on a good processor [12]. The *universe reduction* problem [9] is to bring processors to agreement on a small subset of processors with a fraction of bad processors close to the fraction for the whole set. I.e., the protocol terminates and each good processor outputs the same set of processor ID's such that this property holds. For each of these problems, we say the protocol solves the problem with probability ρ if, given any worst case adversary behavior, including choice of initial inputs, the probability of success of any execution over the distribution of private random coin tosses is at least ρ.

Almost everywhere Byzantine agreement, universe reduction, and leader election is the modified version of each problem where instead of bringing all good processors to agreement, a large majority, but not necessarily all, good processors are brought to agreement.

1.3 Results

We use the phrase *with high probability* (*w.h.p.*) to mean that an event happens with probability at least $1 - 1/n^c$ for every constant c and sufficiently large n. For readability, we treat $\log n$ as an integer throughout. We show:

Theorem 1. [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*. Then, there exists a protocol which w.h.p. computes Byzantine agreement, runs in polylogarithmic time, and uses* $\tilde{O}(n^{3/2})$ *bits of communication.*

This result follows easily from the solution to the universe reduction problem (see the next section) which we present here:

Theorem 2. [Universe reduction] *Let* ϵ *be any positive constant and let* n *be the number of processors in a synchronous fully connected message passing network with a nonadaptive malicious rushing adversary in the full information model which controls less than* $1/3 - \epsilon$ *fraction of processors. For any positive constant* $\epsilon' < \epsilon$*, there exists a protocol which uses* $\tilde{O}(n^{3/2})$ *number of bits of communication per processor and polylogarithmic number of rounds, such that w.h.p., all good processors output the same subset of processors, the "representative set" of size polylogarithmic in* n *such that* $2/3 + \epsilon'$ *fraction of its elements are good.*

1.4 Techniques

Our results build on the almost everywhere universe reduction protocol of [12]:

Theorem 3. *[12]* [Almost everywhere universe reduction] *Let* ϵ *be any positive constant and let* n *be the number of processors in a synchronous fully connected message passing network with a nonadaptive, rushing adversary in the full information model which controls less than* $1/3 - \epsilon$ *fraction of processors. For any positive constant* $\epsilon' < \epsilon$*, there exists a protocol which uses polylogarithmic number of bits of communication per processor and polylogarithmic number of rounds, such that w.h.p.* $1 - O(1/\log n)$ *fraction of good processors output a subset of processors of size polylogarithmic in* n *such that* $2/3 + \epsilon'$ *fraction of its elements are good.*

Our protocol first runs the protocol for almost everywhere universe reduction in [12] to achieve w.h.p. almost everywhere universe reduction. The technical challenge is to go from almost everywhere universe reduction to everywhere universe reduction in $o(n^2)$ bits. It is straightforward to go from everywhere universe reduction to everywhere agreement for Byzantine agreement and leader election (see [12]). The idea is to notice that any "representative" subset of processors can run a standard Byzantine agreement protocol or leader election protocol (using their own input bits, in the case of Byzantine agreement) and the outcome for the representative subset is a solution to the problem for the whole set. The representative set need only communicate its results to the other processors, which determine the correct answer by taking the message sent by the majority.

We actually prove a stronger result than necessary to prove Theorems 1 and 2 from Theorem 3. That is, we can go from almost everywhere universe reduction to everywhere universe reduction even in the case where (1) only $1/2 + \epsilon$ fraction of good processors are in agreement on the representative subset; (2) up to a $1/2 - \epsilon$ fraction of the processors are controlled by the adversary; and (3) communication is in the asynchronous model. Specifically, we show:

Theorem 4. [Almost everywhere to everywhere universe reduction] *Let ϵ be any positive constant and assume n processors are connected in the full information, asynchronous, message passing communication model, with a nonadaptive adversary. Further, suppose there are $(1/2+\epsilon)n$ good processors that agree on a subset C of processors containing a majority of good processors. Then there is a $O(n^{3/2}\log^3 n|C|)$ bit protocol which runs in $O(\log n/\log\log n)$ time steps after which w.h.p. all good processors agree on C.*

We give the Almost Everywhere to Everywhere Universe Reduction Protocol in Section 3; its proof of correctness in Section 4; and include a sketch of the Almost Everywhere Universe Reduction Protocol in the Appendix.

2 Related Work

In a 2006 paper, the authors (and collaborators) present a polylogarithmic time protocol with polylogarithmic bits of communication per processor for almost everywhere Byzantine agreement, leader election, and universe reduction in the synchronous full information message passing model with a nonadaptive rushing adversary [12]. Also in 2006, [8,2] give logarithmic time protocols which use $\Omega(n^2)$ bits of communication for Byzantine agreement in the same model with different techniques. The algorithm in [2] also solves universe reduction and leader election.

In the asynchronous version of the same model, in a 2008 paper [11], the authors give a polynomial time protocol for Byzantine agreement, leader election, and universe reduction. While this protocol uses $\tilde{\Theta}(n^2)$ messages (and polynomial time), its structure is very similar to the almost everywhere agreement protocols [12,13], and we believe it can be implemented as an almost everywhere agreement protocol with polylogarithmic bits of communication.

In the *gossip problem* each process starts with an initial value called a rumor and attempts to learn all the other rumors. In this literature, one concern is the number of messages sent between processors. A 2008 paper [7] presents a protocol to solve the gossip problem in the asynchronous model with crash failures rather than Byzantine failures, with an oblivious adversary which sets the timing and crashes in advance and an assumption of private channels. The protocol in [7] was adapted to solve the consensus problem using $O(n^{7/4}\log^2 n)$ messages. The adversary in [7] is weaker than ours in several respects, though it is stronger in the sense that the adversary can set delays in communication, so our results seem incomparable.

Almost everywhere agreement in sparse networks has been studied since 1986. See [12,13] for references. The problem of almost everywhere agreement for secure multiparty computation on a partially connected network was defined and solved in 2008 in [6].

In a 2006 paper [13], the authors give a sparse network implementation of their protocols from [12]. It is easy to see that everywhere agreement is impossible in a sparse network where the number of faulty processors t is sufficient to surround a good processor. To see this, one can use an observation from [10]. Let t be the number of bad processors. Then any Byzantine agreement protocol where all $n-t$ good processors have their input bits set to 1 must result in an output of

1. And this must be true even if the bad processors act like good processors that have a 0. Moreover, it must be the case that when bad processors act like good processors that have a 1 and t or fewer good processors have a 0, the output must be a 1 as well. If a processor is surrounded by bad processors, then all communication with the processor can be made to simulate any execution of the protocol consistent with that processor's input bit. Hence if a single processor has an input bit of 0, and it is surrounded by bad processors, it will be unable to distinguish between the case where it must output a 0 because all good processors have a 0, or a 1 because fewer than t processors have a 0.

A protocol in which processors use $o(n)$ bits may seem as vulnerable to being isolated as in a sparse network, but the difference is that without access to private random bits, the adversary can't anticipate at the start of the protocol where communication will occur. In [10], it is shown that even with private channels, if a processor must pre-specify the set of processors it is willing to listen to at the start of a round, where its choice in each round can depend on the outcome of its random coin tosses, at least one processor must send $\Omega(n^{1/3})$ messages to compute Byzantine agreement with probability at least $1/2 + 1/\log n$. Hence the only hope for a protocol where every processor sends $o(n^{1/3})$ messages is to design outside this constraint. Note that the protocol here does NOT fall within this restrictive model, only because of line 8 in our Almost Everywhere to Everywhere protocol, where the decision of whether a message is listened to (or acted upon) depends on how many messages are received so far.

3 The Almost Everywhere to Everywhere Universe Reduction Protocol

In this section, we describe the algorithm that satisfies Theorem 4 by going from almost everywhere committee election to everywhere committee election.

Precondition: Each processor p starts with an hypothesis of the membership of C, C_p; this hypothesis may or may not be equal to C or may be empty. However, the following two assumptions are critical. First, there exists a subset of the processors, C, of polylogarithmic size, with a majority of good processors. Second, there is a set S of at least $(1/2 + \epsilon)n$ good processors, such that for all $p \in S$, $C_p = C$.

Overview of Algorithm: The main idea of this protocol is for each processor p to randomly select $c \log n$ processors to poll as to the membership of C. Unfortunately, if these requests are made directly from p, the adversary can flood the network with "fake" requests so that the good processors are forced to send too many responses. Thus, the polling request are made through the set C, which counts the messages received from each processor to enforce that that total number of polling requests sent out is not too large.

Unfortunately, this approach introduces a new problem: processor p may have an incorrect guess about the membership of C. We solve this by having p send

a (type 1) message containing its poll-list ($Poll_p$) to $List_p$, a set of $c \log n\sqrt{n}$ randomly sampled processors. Processor p hopes that at least one processor in the set $List_p$ will have a correct guess about C and will thus be able to forward a (type 2) message containing $Poll_p$ to C. To prevent these processors $q \in List_p$ from being flooded, each such processor q only forwards a type 2 message from a processor p if p appears in the set $Forward_q$, which is a set of $\sqrt{n}$ processors that are randomly sampled in advance. Upon receiving a $< Poll_p, p >$ (type 2) message from any processor q, a processor in C then sends a (type 3) request with p's ID to each member $s \in Poll_p$. More precisely, a processor in C only processes the first $\sqrt{n}$ such type 2 messages that it receives from any given processor q: this is the crucial filtering that ensures that the total number of requests answered is not too large. Upon receiving a type 3 request, $< p, 3 >$ from a majority of C, s sends C_s to p, a (type 4) message.

There are two remaining technical problems. First, since a confused processor, p, can have a C_p equal to a mostly corrupt set C', C' can overload every confused processor. Hence we require that any processor, p, who receives an overload (more than $\sqrt{n}\log^2 n$) of type 3 requests wait until their own C_p is verified before responding. Second, the processors in C handle many more requests than the other processors. The adversary can conceivably exploit this by bombarding confused processors which think they are in C with type 2 requests. Thus, the algorithm begins with a verification of membership in C. Each processor p sends a request message to a randomly selected sample ($Poll_p$) which is responded to by a polled processor q if and only if $p \in C_q$.

Example: An example run of our algorithm is shown in Figure 1. This figure follows the technically challenging part of our protocol, steps 6-10, which are

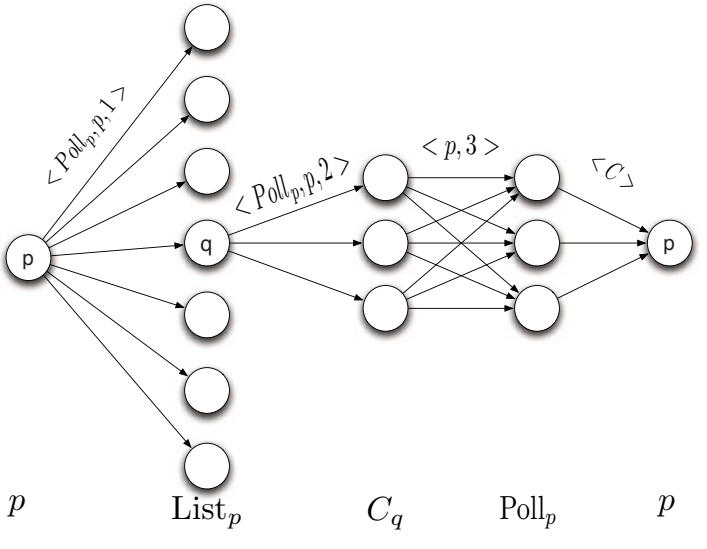

Fig. 1. Steps 6-10 of Our Protocol

described in detail in Algorithm 1 listed below. In Figure 1, time increases in the horizontal direction. This figure concerns a fixed processor p that concludes $p \notin C$ in the earlier parts of the algorithm (steps 2-5). For clarity, in this example, only messages that are sent on behalf of p that eventually help p to determine C are shown. Moreover, again for clarity, we show a best case scenario where all nodes in $Poll_p$ are assumed to have received no more than $\sqrt{n}\log^2 n$ type 3 requests. In the first step of this example, p sends the message $< Poll_p, p, 1 >$ to all nodes in $List_p$. The node q is the only node in this set such that $p \in Forward_q$, so q forwards a type 2 message of the form $< Poll_p, p, 2 >$ to all the nodes in C_q. In this example, $C_q = C$. Next all nodes in C_q send the message $< p, 3 >$ to all nodes in $Poll_p$. In this example, all nodes in $Poll_p$ know the set C, so they all send the message $< C >$ to p in the final step.

Algorithm 1. Almost Everywhere to Everywhere

Each processor executes the following steps in any order:

1. Each processor p selects uniformly at random, independently, and with replacement three subsets, $List_p$, $Forward_p$, and $Poll_p$ of processor ID's where: $|List_p| = c\sqrt{n}\log n$; $|Forward_p| = \sqrt{n}$; $|Poll_p| = c\log n$;

Verifying Membership in C:

2. $member_p \leftarrow FALSE$
3. If $p \in C_p$, then p sends a message $< Am\ I\ in\ C? >$ to the members of $Poll_p$;
4. If q receives a message $< Am\ I\ in\ C? >$ from a processor $p \in C_q$, q sends $< Yes >$ back to the p;
5. If p receives a message $< Yes >$ from a majority of members of $Poll_p$ then p sets $member_p \leftarrow TRUE$;

Determing C:

6. p sends a message $< Poll_p, p, 1 >$ (type 1 message) to each processor in $List_p$;
7. For each q: if $< Poll_p, p, 1 >$ is the first type 1 message received from processor p and $p \in Forward_q$, then q sends $< Poll_p, p, 2 >$ (a type 2 message) to every processor in C_q;
8. For each r: if $member_r = TRUE$ then for every processor q, for the first $\sqrt{n}$ type 2 messages of the form $< Poll_p, p, 2 >$ which are received from q, send $< p, 3 >$ (type 3 message) to every processor in $Poll_p$;
9. For each s: for the first $\sqrt{n}\log^2 n$ different type 3 messages of the form $< p, 3 >$ which are each sent by a majority of processors in C_s, send $< C_s, 4 >$ (type 4 message) to p;
10. If s receives the same type 4 message $< C', 4 >$ from a majority of processors in $Poll_s$ then
    (a) s sets $C_s \leftarrow C'$; and
    (b) s answers any remaining type 3 requests that have come from a majority of the current C_s, i.e. for each such request $< p, 3 >$ s sends $< C_s, 4 >$ to p;

4 Proof of Correctness

First, we point out that the asynchronicity of the model is not a real problem here because of the following observation:

Observation 1. *In the asynchronous model, if p is waiting to hear from a set of processors such that a majority of processors in the set are good and agree on the same value, and if each sends that value to p, then the adversary cannot prevent p from receiving this value in one timestep.*

In what follows, we show that with high probability, all transmissions which processors need to respond to are sent by a majority of good processors which agree on the same value. We say that a processor is *knowledgeable* if it is good and $C_p = C$.

Lemma 1. *W.h.p., more than a* $1/2+\epsilon/2$ *fraction of processors of every poll-list are knowledgeable at the start of the protocol and these remain knowledgeable.*

Proof. Let c' be any positive constant and c be the constant in the protocol. Let X be the number of processors which are initially knowledgeable in a fixed poll-list. Then $E[X] = (1/2 + \epsilon)c \log n$. The probability that the number of initially knowledgeable processors on the poll-list is not a majority is less than the probability that $X \leq (1-\delta)E[X]$ for $\delta = (\epsilon/2)/(1/2+\epsilon)$. Using the Chernoff bound, this probability is $\leq e^{(-\delta^2 E[X]/2)} \leq n^{-c'-1}$ for $c = 8(c'+1)(1/2+\epsilon)/\epsilon^2$, i.e., for c a constant, this is $1/n^{c'+1}$.

There are no more than n poll-lists for good processors. Thus, the probability that any poll-list fails to have at least a $1/2 + \epsilon/2$ fraction of initially knowledgeable processors is no greater than the sum of the n individual probabilities of failure or $1/n^{c'}$ for any constant c'.

Next, we prove by contradiction that no knowledgeable processor becomes not knowledgeable. Let p be the first processor which becomes not knowledgeable. This implies that p resets $C_p \neq C$, which implies that $Poll_p$ contains less than a majority of knowledgeable processors. By assumption, the initially knowledgeable processors in $Poll_p$ are still knowledgeable, which implies there must have been less than a majority of initially knowledgeable processors in $Poll_p$. But we have shown this event does not occur w.h.p. for any poll-list.

Lemma 2. *W.h.p., if every type 3 message received by every knowledgeable processor p is responded to by p within* $O(\log n/\log\log n)$ *steps then Theorem 4 holds. Moreover, a total of* $O(n^{3/2}\log^3 n|C|)$ *bits are sent.*

Proof. We fix a good processor p and first bound the probability that $Poll_p$ is forwarded to C. The probability that a knowledgeable processor q forwards a type 1 message sent to it by a good processor p is the probability that $p \in Forward_q = 1/\sqrt{n}$. Since each processor p sends out $c\sqrt{n}\log n$ type 1 messages, the probability that all fail to be forwarded is the probability that for every message, the processor q receiving it is either not knowledgeable or q is

knowledgeable but $p \notin Forward_q$. Recall that by assumption, initially no more than $1/2 - \epsilon$ fraction of processors are not knowledgeable and by the previous lemma, this number does not grow. The probability that $Poll_p$ is not forwarded to C by a particular processor q which receives it is therefore bounded above by $(1/2 - \epsilon) + (1/2 + \epsilon)(1 - 1/\sqrt{n}) = 1 - \frac{1/2+\epsilon}{\sqrt{n}}$. The probability that all requests to forward fail is $(1 - (1/2 + \epsilon)/\sqrt{n})^{c\sqrt{n}\log n} \leq 1/n^{c/2}$.

If $Poll_p$ is forwarded by a knowledgeable processor then it is sent to every processor in C, by the definition of knowledgeable. From the previous lemma, and a simple Chernoff and union bound, each processor in C verifies it is in C with high probability. Thus, since a good processor never forwards more than $\sqrt{n}$ messages, with high probability all good processors in C will send a message of the form $< p, 3 >$ to each processor in $Poll_p$. In particular, a majority of processors in C will do so. With high probability (from the previous lemma) a majority of processors in $Poll_p$ are knowledgeable and hence upon receiving messages from processors in C will send p the correct message which p receives and decides correctly. Taking the union over all processors p, for any constant c', there is a constant c for the algorithm such that the probability of any processor failing is no greater than $1/n^{c'}$.

The number of bits transmitted by good processors can be calculated as follows. First we consider bits sent by processors that are knowledgeable. For the verification phase, the total number of bits is $\tilde{O}(n \log n)$. For the next phase, each poll-list contains $O(\log n)$ ID's of $O(\log n)$ length for a total of $O(\log^2 n)$ bits. Each good processor forwards no more than $\sqrt{n}$ poll-lists to the members of C for a total of $O(n^{3/2}|C| \log^2 n)$ bits transmitted. With high probability, each member of C transmits $O(n^{3/2} * |poll - list|)$ messages each with at most $O(\log n)$ bits per message to each member of each poll-list in the form of type 3 messages, for $O(n^{3/2}|C| \log^2 n)$ bits in total. The knowledgeable processors which receive type 3 messages respond to them all, for a total of $O(n^{3/2}|C| \log^2 n)$ bits. The total number of bits transmitted by knowledgable processors is thus $O(n^{3/2}|C| \log^2 n)$. In addition, no more than $n(\sqrt{n} \log^2 n)$ type-4 messages are sent by processors while they are not knowledgeable, for a total of $n(\sqrt{n} \log^3 n|C|)$ bits.

It remains to show:

Lemma 3. *W.h.p., every type 3 message received by every knowledgeable processor p is responded to by p within $O(\log n / \log \log n)$ steps.*

Proof. A knowledgeable processor is *overloaded* if there are more than $\sqrt{n} \log^2 n$ poll-lists received by C which contain it. As there are no more than $n^{3/2}$ type 2 messages processed by C and each poll-list has size $c \log n$, there can be no more than $cn/\log n$ processors which receive more than $\sqrt{n} \log^2 n$ type 3 requests from C.

The adversary can choose its poll lists after seeing the poll lists from all the good processors. We will denote time step i of our algorithm to begin when i units of time have elapsed and end just before $i + 1$ units have elapsed, where a unit of time is defined to be the maximum delay of a message. We will say

$Know_p = FALSE$ at time i if p has not yet received the same type 4 message from a majority of processors in $Poll_p$, i.e. the condition for the if statement in step 10 of our algorithm has not been satisfied, by time i. Otherwise, we will say $Know_p = TRUE$. A processor p is *blocked* at time i if $Know_p = FALSE$ and p is overloaded.

Claim: With high probability, for any time step $j \geq 6$, if there is a processor with $Know_p = FALSE$ at time step $i + j$, then there must be $((\epsilon c \log n)/4)^i (\log \log n)^{i-1}$ distinct blocked processors at time j, for $i = 1, ..., O(\log n / \log \log n)$.

Proof of Claim: Let $L = |poll - list| = c \log n$. We note that since $j \geq 6$, by time step j, w.h.p., all type 1, 2 and 3 messages have been sent out and received. Moreover, the first set of type 4 messages have been sent and received. This proof is by induction on i.

Fix a processor p. Then we can view p as a root of a tree. Each node is a processor; the children of each node q are the processors in $Poll_q$. Note that some processors may appear more than once in the tree. The degree of each node is L.

Base Case: For $i = 1$. Suppose there are fewer than $(\epsilon/4)L$ blocked processors in time step j. Then from Lemma 1, w.h.p., there are $(1/2+\epsilon/2)L$ processors on every poll-list which are knowledgeable. Then there remain $(1/2 + \epsilon/4)L$ knowledgeable processors on $Poll_p$ who are not blocked and will send type 4 messages to p in the next timestep. In time $j + 1$, p will hear from them and decide.

Induction Step: Let $x_i = (\epsilon L/4)^i (\log \log n)^{i-1}$. Assume the induction hypothesis holds for $i - 1$. Then if there is a processor with $Know_p = FALSE$ at time $i + j$, then there must be a set S of size x_{i-1} of blocked processors at time step $j + 1$. Then it must be the case that at time j, reasoning as in the base case, each element of S must have at least x_1 blocked children (i.e., elements of its poll-list). We show that w.h.p. there is no set S' of size less than x_i which satisfies this condition.

Fix a set S, a set of x_1 children for each element of S, and a set S'. Since the children are picked randomly and independently, the probability of having x_1 children for each element of S coming from S' is

$$(x_i/n)^{x_1 x_{i-1}} \leq (x_i/n)^{x_i \log \log n}.$$

The number of ways to choose these sets is no more than

$$\binom{n}{x_{i-1}} L^{x_1 x_{i-1}} \binom{n}{x_i} = (ne/x_{i-1})^{x_{i-1}} L^{x_1 x_{i-1}} (ne/x_i)^{x_i} \leq (ne/x_i)^{cx_i};$$

where the last inequality holds since we can assume that $x_i \leq n/2$. Taking the union bound over all possible such sets, we find that the probability of there existing a set S' is less than

$$(x_i/n)^{(\log \log n - O(1))((\epsilon L/4)/\log \log n)^i} < 1/n^{c'}$$

for any constant c' and $x_i/n \leq n/\log n$. Taking the union bound over all i yields the claim.

Remaining Proof of Lemma: For $i = O(\log n/\log\log n)$, the required number of blocked processors exceeds $cn/\log n$, the maximum number the adversary is able to block. Hence, every processor decides by time $O(\log n/\log\log n)$.

5 Conclusion and Open Problems

We have shown that classical problems in distributed computing, like Byzantine agreement, universe reduction, and leader election, can be solved with high probability using only $\tilde{O}(n^{3/2})$ bits of communication, even if the adversary has unlimited resources.

Several open problems remain including the following. First, we believe that the protocol from [12] for electing a committee that contains a $2/3+\epsilon$ fraction of good processors using polylogarithmic bits per processor can be made to work in the asynchronous model. This would imply, together with the results in this paper that universe reduction, Byzantine agreement, and leader election could all be performed with $\tilde{O}(n^{3/2})$ bits in the asynchronous model.

Second, we conjecture that the number of bits required for Byzantine agreement in the full information model with a nonadaptive adversary is $\Omega(n^{3/2})$ in both the synchronous and asynchronous models unless a superpolylogarithmic time is incurred. Third, we ask: Is there is a load-balanced version of the protocol presented here in which each processor needs to send only $\tilde{O}(\sqrt{n})$ bits? Fourth, we ask: Can this bound be beaten if cryptographic assumptions are incorporated into the model? Finally, can other problems like secure mulitparty computation be solved with $o(n^2)$ bits of communication?

Acknowledgements

The authors would like to thank the program committee for a careful, extensive review.

References

1. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edn. John Wiley Interscience, Chichester (2004)
2. Ben-Or, M., Pavlov, E., Vaikuntanathan, V.: Byzantine agreement in the full-information model in o(log n) rounds. In: STOC, pp. 179–186 (2006)
3. Bortnikov, E., Gurevich, M., Keidar, I., Kliot, G., Shraer, A.: Brahms: Byzantine resilient random membership sampling. In: PODC 2008: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing, Toronto, Canada, pp. 145–154. ACM, New York (2008)

4. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. J. ACM 32(1), 191–204 (1985)
5. Feige, U.: Noncryptographic selection protocols. In: FOCS 1999: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, p. 142. IEEE Computer Society, Washington (1999)
6. Garay, J.A., Ostrovsky, R.: Almost-everywhere secure computation. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 307–323. Springer, Heidelberg (2008)
7. Georgiou, C., Gilbert, S., Guerraoui, R., Kowalski, D.R.: On the complexity of asynchronous gossip. In: Proceedings of the ACM symposium on Principles of distributed computing (PODC), pp. 135–144 (2008)
8. Goldwasser, S., Pavlov, E., Vaikuntanathan, V.: Fault-tolerant distributed computing in full-information networks. In: FOCS, pp. 15–26 (2006)
9. Gradwohl, R., Vadhan, S.P., Zuckerman, D.: Random selection with an adversarial majority. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 409–426. Springer, Heidelberg (2006)
10. Holtby, D., Kapron, B.M., King, V.: Lower bound for scalable byzantine agreement. Distributed Computing 21(4), 239–248 (2008)
11. Kapron, B.M., Kempe, D., King, V., Saia, J., Sanwalani, V.: Fast asynchronous byzantine agreement and leader election with full information. In: SODA, pp. 1038–1047 (2008)
12. King, V., Saia, J., Sanwalani, V., Vee, E.: Scalable leader election. In: SODA 2006: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 990–999. ACM Press, New York (2006)
13. King, V., Saia, J., Sanwalani, V., Vee, E.: Towards secure and scalable computation in peer-to-peer networks. In: FOCS 2006: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 87–98. IEEE Computer Society, Washington (2006)
14. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. SIGOPS Oper. Syst. Rev. 41(6), 45–58 (2007)

Appendix: Sketch of Almost Everywhere Universe Reduction

Here, we include a sketch of the protocol to compute almost everywhere universe reduction, excerpted from [12]. The processors are assigned to groups of polylogarithmic size; each processor is assigned to multiple groups. In parallel, each group then elects a small number of processors from within their group to move on. We then recursively repeat this step on the set of elected processors until the number of processors left is polylogarithmic.

The method used to run elections is a simple adaptation from the atomic broadcast model to the synchronous distributed model of a subroutine in [5]:

ELECT-SUBCOMMITTEE: *Input is processors* $p_1, \ldots, p_k$
with $k = \Omega(\ln^8 n)$.
1 For $i = 1$ to k,
2 Processor p_i randomly selects one of $k/(c_1 \ln^3 n)$

"bins" and tells the other processors in its committee which bin it has selected.
3 The other processors in the committee run Byzantine Agreement to come to a consensus on which bin p_i has selected.
4 Let B be the bin with the least number of processors in it, and let S_B be the set of processors in that bin. Arbitrarily add enough processors to S_B to ensure $|S_B| = c_1 \ln^3 n$.
5 Return S_B as the elected subcommittee.

Although this approach is intuitively simple, there are several complications that must be addressed.

(1) The groups must be determined in such a way that the election mechanism cannot be sabotaged by the bad processors.
(2) After each step, each elected processor must determine the identities of certain other elected processors, in order to hold the next election.
(3) Election results must be communicated to the processors.
(4) To ensure load balancing, a processor which wins too many elections in one round cannot be allowed to participate in too many groups in the next round.

Item (1): we use a layered network with extractor-like properties. Every processor is assigned to a specific set of nodes on layer 0 of the network. In order to assign processors to a node A on layer $\ell > 0$, the set of processors assigned to nodes on layer $\ell - 1$ that are connected to A hold an election. In other words, the topology of the network determines how the processors are assigned to groups. By choosing the network to have certain desired properties, we can ensure that the election mechanism is robust against malicious adversaries.

To accomplish item (2), we use *monitoring sets*. Each node A of the layered network is assigned a set of nodes from layer 0, which we denote $m(A)$. The job of the processors from $m(A)$ is simply to know which processors are assigned to node A. Since the processors of $m(A)$ are fixed in advance and known to all processors, any processor that needs to know which processors are assigned to A can simply ask the processors from $m(A)$. (In fact, the querying processor only needs to randomly select a polylogarithmic subset of processors from $m(A)$ in order to learn the identities of the processors in A with high probability. This random sampling will be used to ensure load balancing.)

Since the number of processors that need to know the identities of processors in node A is polylogarithmic, the processors of $m(A)$ will not need to send too many messages, but they need to know which processors need to know so they do not respond to too many bad processors' queries. Hence the monitoring sets need to inform relevant other monitoring sets of this information.

Item (3): We use a *communication tree* connecting monitoring sets of children in the layered networks with monitoring sets of parents to inform the monitoring sets which processors won each of their respective elections and otherwise pass information to and from the individual processors on layer 0.

Item (4) is addressed by having such processors refrain from further participation.

The protocol results in almost everywhere agreement rather than everywhere agreement, because the adversary can control a small fraction of the monitoring sets by corrupting their nodes. Thus communication paths to some of the nodes are controlled by the adversary.