

Distributed Agreement with Optimal Communication Complexity

Seth Gilbert
EPFL
seth.gilbert@epfl.ch

Dariusz R. Kowalski
University of Liverpool
D.Kowalski@liverpool.ac.uk

Abstract

We consider the problem of fault-tolerant agreement in a crash-prone synchronous system. We present a new randomized consensus algorithm that achieves optimal communication efficiency, using only $O(n)$ bits of communication, and terminates in (almost optimal) time $O(\log n)$, with high probability. The same protocol, with minor modifications, can also be used in *partially synchronous* networks, guaranteeing correct behavior even in asynchronous executions, while maintaining efficient performance in synchronous executions. Finally, the same techniques also yield a randomized, fault-tolerant gossip protocol that terminates in $O(\log^* n)$ rounds using $O(n)$ messages (with bit complexity that depends on the data being gossiped).

1 Introduction

Fault-tolerant agreement, also known as consensus, is perhaps one of the most fundamental problems in distributed computing. The problem of consensus, first introduced in 1980 [25, 22], is formally defined as follows:

DEFINITION 1. (CONSENSUS) *Given n processes, at most t of which may crash: each process p_i begins with initial value $v_i \in \{0, 1\}$ and decides on an output satisfying: (1) Agreement: every process decides the same value; (2) Validity: if a process decides v , then v is some process's initial value; (3) Termination: every correct process eventually decides, with probability 1.*

In this paper, we assume that the decision is binary; generalizations for any input value domain are straightforward, though the communication cost scales with the size of the input values.

The Question of Efficiency. In this paper, we address the question of how efficiently agreement can be achieved. There are two basic metrics: *time complexity* and *communication complexity*. Many existing protocols achieve optimal time complexity. For example, the FLOODSET algorithm [12, 23] terminates in $t+1$ rounds, which is optimal for deterministic algorithms. Chor et al. [11] develop a randomized algorithm that takes

$O(\log n)$ rounds, with high probability, which is almost optimal (under an oblivious adversary); they also show a lower bound of $\Omega(\log n / \log \log n)$ rounds to achieve agreement, with high probability.

Unfortunately, these protocols have relatively high communication complexity, as in each round, each process broadcasts a message to every other process, resulting in $\Theta(n^2)$ messages per round. By contrast, solving consensus requires at least $\Omega(n)$ bits of communication, as every process needs to either send or receive one message. In fact, Amdur et al. [1] showed that even in a failure-free execution, $\Omega(n)$ message complexity is needed. This leaves a significant gap between the upper and lower bounds.

There has been significant progress in addressing this longstanding question (see Figure 1 for key results). Dwork et al. [13] made an important breakthrough: a deterministic consensus protocol with $O(n \log n)$ message complexity and exponential time complexity. Galil et al. [15] improved on this with an algorithm using $O(n)$ messages and superlinear time $O(n^{1+\epsilon})$, for any $0 < \epsilon < 1$. The best *linear-time* deterministic algorithm to-date is by Chlebus and Kowalski [6, 7], and has $O(n \log^{O(1)} n)$ message complexity, but $\Omega(n^2)$ communication complexity. Chlebus et al. [10] have recently improved on this, developing a linear-time deterministic algorithm with $O(n \log^4 n)$ communication complexity.

Chlebus and Kowalski [8] have also recently developed a new *randomized* algorithm that, with high probability, takes $O(\log n)$ time and $O(n \log n)$ communication complexity (subject to an oblivious adversary). A key novelty of this protocol is that it is *local*, i.e., no process sends more than $O(\log n)$ bits (using entirely different techniques than in this paper).

Main Result. In this paper, we present the first consensus protocol to achieve both optimal communication complexity and almost optimal time complexity, while tolerating up to $f < n/2$ crash failures. Specifically, we present a randomized consensus protocol that, with high probability, has both $O(n)$ communication complexity and $O(\log n)$ round complexity.

Surprisingly, our algorithm is, perhaps, simpler than many earlier algorithms, relying on the technique of *universe reduction* to eliminate much of the complexity while still achieving efficient performance. (See discussion below for earlier papers on universe reduction.) Instead of solving consensus among all n nodes, which is potentially expensive, we delegate the work of agreement to $\Theta(\log n)$ special *coordinators*, chosen at random. These coordinators then execute an existing consensus protocol, and distribute the results. Selecting these coordinators, maintaining their consistency, and managing the data dissemination are the main challenges that arise in implementing this approach:

Coordinator discovery: Once the coordinators have been randomly selected, each must efficiently discover the others, without sending too many messages. (The trivial solution in which each of the $\Theta(\log n)$ coordinators sends a message to every other process announcing its selection requires $\Theta(n \log n)$ messages.) Moreover, there is some (perhaps small) probability that the discovery process fails, resulting in several disconnected cliques of coordinators. The discovery protocol is engineered to cope with these problems.

Coordinator consistency: As the protocol proceeds, some of the coordinators may fail, by crashing. The protocol must maintain the consistency of the coordinators, ensuring that they have a similar (though not necessarily identical) view of the system as they progress through the protocol.

Data dissemination: The coordinators must efficiently work together to disseminate the decision to the other processes. (Again, the trivial solution in which each coordinator sends the decision to every process requires $\Theta(n \log n)$ messages.) In addition, some of the coordinators may fail during the protocol. We rely on a fault-tolerant work-sharing paradigm in which the coordinators repeatedly exchange information to prevent wasted work. The resulting simple randomized process takes only $O(\log^* n)$ rounds and $O(n)$ messages to disseminate a piece of information.

These challenges must be overcome with a minimum of communication. Since we are attempting to achieve $O(n)$ communication complexity, messages must be kept to a minimum. For example, many consensus protocols rely on leader election; however, we cannot even afford to distribute the identity of a leader to every process in the system as that would require $O(n)$ messages each of size $\log n$ bits.

Secondary Results. A natural question is whether agreement can also be efficiently achieved in a *partially synchronous* network, i.e., a network that is almost always synchronous, but occasionally suffers from unpredictable message delays. Previously [17], we showed

how to transform protocols for synchronous systems into protocols for partially synchronous systems. A similar idea can be used here. Our techniques yield a protocol that achieves $O(n)$ communication complexity when the network is synchronous, and yet still operates correctly even when the network is asynchronous.

Another problem of significant recent interest in distributed computing is *gossip*: each process in the system starts with a *rumor*; the goal, in the end, is for every process to learn every rumor. It is well-known that in a synchronous system, a simple randomized rumor-spreading process completes in $O(\log n)$ rounds with $O(n \log n)$ message complexity. In an important paper, Karp et al. [19] studied a more intricate “push-pull” randomized process which achieves $O(n \log \log n)$ message complexity in $O(\log n)$ rounds (and showed that among a certain class of algorithms, this was optimal). Using the *universe reduction* technique, along with the coordinator protocols described in this paper, we show how to achieve *fault-tolerant* gossip in a synchronous network in only $O(\log^* n)$ rounds, using only $O(n)$ messages, with high probability¹.

Other Related Work. There has been much important research developing consensus protocols (and lower bounds) for other adversarial models, including asynchronous networks and networks subject to Byzantine (malicious) failures (see [2] for further references). Of note, recent work by Attiya et al. [4, 3] has (effectively) resolved the question of work-optimal consensus in an asynchronous shared memory. Also of note, Georgiou et al. [16] developed the first consensus algorithm for asynchronous networks that achieved sub-quadratic (but super-linear) message complexity.

The technique of *universe reduction* has been an important tool in developing distributed algorithms; we focus here on prior work related to consensus. Particularly notable are randomized algorithms by Ben-Or et al. [5] and Kapron et al. [18] that use universe reduction to solve Byzantine agreement (i.e., in the presence of malicious participants) in $O(\log n)$ expected time, but with relatively high message complexity (of $\Omega(n^2)$). Using similar ideas, King et al. [21] show how to elect an honest leader using only $O(n \log^{O(1)} n)$ bits of communication. Quite recently, King and Saia [20] have built on their leader election protocol to solve Byzantine agreement (with a probabilistic agreement guarantee) using only $\tilde{O}(n^{3/2})$ bits of communication;

¹The communication complexity, however, depends on the size of the rumors. Often, however, the goal of gossip is to calculate some aggregate information, for example, the average of all the rumors. In this case, the bit complexity can be kept small. Notice that the same problem of message size arises in all random rumor spreading protocols.

	Message/Bit Complexity	Round Complexity	Randomized?
FloodSet	$O(n^3)$ bits	$O(n)$	No
Optimized FloodSet	$O(n^2)$ bits	$O(n)$	No
GMV'95 [15]	$O(n)$ messages	$O(n^{1+\epsilon})$	No
GMV'95 [15]	$O(n)$ bits	$2^{O(n)}$	No
CK'02,CK'06 [6, 7]	$O(n \log^{O(1)} n)$ messages	$O(n)$	No
CKS'09 [10]	$O(n \log^{O(1)} n)$ bits	$O(n)$	No
CMS'89 [11]	$O(n^2 \log n)$ messages	$O(\log n)$	Yes
CK'09 [8]	$O(n \log n)$ bits	$O(\log n)$	Yes
Section 3	$O(n)$ bits	$O(\log n)$	Yes

Figure 1: Summary of communication complexity and round complexity of existing consensus protocols compared to the new algorithm presented in this paper. The randomized bounds hold with high probability.

this is the first such protocol that breaks the quadratic communication complexity barrier!

In this paper, we apply the technique of universe reduction to a system subject to crash failures, not Byzantine failures. Thus some aspects of our protocol are simpler, since there is no need to prevent Byzantine participants from hijacking the selected sub-universe. On the other hand, some aspects are more complicated as careful coordination is needed to reach $O(n)$ communication complexity. As one example of the challenges, we cannot even afford the cost of informing everyone as to who is in the selected sub-universe.

2 Model

We consider a system consisting of n processes $\Pi = p_1, p_2, \dots, p_n$. Up to $f < n/2$ processes may fail by crashing. An execution is divided into synchronous rounds, and processes communicate by exchanging messages in each round. If a process does not fail by the end of a round, then all of the messages that it sends in that round are delivered. On the other hand, if a process fails during a round, then an arbitrary subset (possibly all) of its messages for that round are lost, as determined by the adversary. The *message complexity* of an execution is the total number of messages sent by all processes during the entire execution. The *communication complexity* of an execution is the total number of bits for all messages.

Each process has access to an arbitrary number of random bits. We assume that the round in which a process fails is independent of the random bits, i.e., the adversary is *oblivious*. Formally, the oblivious adversary

chooses, prior to every execution: (i) a function $F : \Pi \rightarrow \mathbb{Z} \cup \{\perp\}$ specifying in which round each process fails, and (ii) a function $M : \Pi \rightarrow \mathbb{P}(\Pi)$ specifying which subset of messages are delivered by a process in the round in which it fails. If a process p never fails, i.e., if $F(p) = \perp$, we say that it is *correct*. We say that a process p is *non-failed* in round r if it is either correct or $F(p) > r$. If a message is sent by a faulty process p in round $F(p)$ to a process $q \in M(p)$, then it is delivered; if it is sent to a process $q' \notin M(p)$ then it is not delivered²

3 Communication-Optimal Consensus

In this section, we present a randomized algorithm for solving consensus using only $O(n)$ messages and $O(n)$ bits of communication. We begin with an overview of the basic structure of the algorithm. We then describe the individual building blocks, and discuss how to assemble them into an efficient solution.

The first step, described in Section 3.1, is to choose a set of $\Theta(\log n)$ coordinators that are responsible for determining the eventual decision. The coordinators are selected at random, and then participate in a simple discovery process to find the other coordinators. (This discovery process can be viewed as an example of *probabilistic quorum systems* [24].) The second step, described in Section 3.2, is for the coordinators to run a consensus protocol amongst themselves; we refer to this as a *limited universe consensus protocol*, as only

²Note that the protocols in this paper can in fact tolerate a somewhat stronger adversary, in that it is only necessary that the adversary decide in advance which processes will fail; it can adaptively choose when and how they fail.

the coordinators participate. Since there are very few coordinators, this can be implemented quite efficiently. The third step, described in Section 3.3., is for the coordinators to disseminate the decision value to the remaining processes. If something goes wrong during any of these steps, the processes abort and execute a fall-back consensus protocol that is less efficient. We now describe each of the components of the protocol in more detail. Throughout, we fix a constant c that is used in the probabilistic analysis.

3.1 Choosing Coordinators. The CHOOSEC sub-protocol is responsible for selecting a set of coordinators. It is a probabilistic protocol that guarantees, with high probability, that when the protocol completes there are $\Theta(\log n)$ coordinators, each aware of the entire set of coordinators. More specifically, for each process p_i , the sub-protocol returns two outputs: (1) a boolean flag isC_i indicating whether process p_i is a coordinator, and (2) a list $coords_i$ of other coordinators. The sub-protocol guarantees the following properties at the end of the execution:

- *Self-Inclusion^{CC}*: For every process p_i , if $isC_i = \text{TRUE}$, then $p_i \in coords_i$.
- *Probabilistic Uniformity^{CC}*: With high probability, there exists a subset S such that: (i) every process $p_i \in S$ is a coordinator ($isC_i = \text{TRUE}$); (ii) every non-failed coordinator at the end of the CHOOSEC protocol is a member of S ; (iii) for each non-failed coordinator $p_i \in S$, $coords_i \subseteq S$; (iv) for each non-failed coordinator $p_i \in S$, for each process $p_j \in S \setminus coords_i$, process p_j fails by the end of the CHOOSEC protocol.
- *Coordinator Set Size^{CC}*: With high probability, when the protocol completes, the set of correct coordinators has size $\Theta(\log n)$.
- *Termination^{CC}*: The protocol completes in $O(1)$ rounds.

The probabilistic uniformity condition captures the intuition that the set $coords_i$ at process p_i reflects exactly the set of coordinators; however different processes may have slightly different views of which processes are coordinators, due to the fact that some candidate “coordinators” may fail during the protocol. Notice that both the *probabilistic uniformity* and *coordinator set size* properties hold with high probability. By contrast, the *self-inclusion* and *termination* properties hold with probability 1.

Sub-protocol description. The CHOOSEC sub-protocol consists of the following steps:

Round 0. Initially, each process p_i independently sets isC_i to TRUE with probability $8c \log n/n$. From this point on, we refer to the processes that set isC to TRUE in this step as *coordinators*.

Round 1. Each coordinator chooses a set of $\Theta(\sqrt{n} \log n)$ *intermediaries* (specifically, $2c\sqrt{n} \log n$ intermediaries, for some constant c). Coordinator p_i sends a message to each selected intermediary containing the identifier “ p_i .” Each such message is of size $\log n$ bits.

Round 2. Each intermediary sends a single response to each message that it received in round 1. The response of an intermediary p_j contains all the messages received by p_j in round 1. Each coordinator p_i combines the responses it receives from intermediaries, along with its own identifier, to form the list $coords_i$.

Analysis. We first bound the size of the set of coordinators, showing that with high probability, the set of correct coordinators is $\Theta(\log n)$. Let C_0 be the set of processes that set $isC = \text{TRUE}$ in Round 0. Let CORRECT be the set of correct processes that never fail (even after the CHOOSEC protocol completes).

LEMMA 3.1. (COORDINATOR-SET SIZE) *With probability at least $1 - 1/n^c$: (i) $|C_0| \leq 16c \log n$; and (ii) $c \log n \leq |C_0 \cap \text{CORRECT}|$.*

Proof. Let X_i be a 0/1 variable indicating whether $isC_i = \text{TRUE}$, and recall that $|\text{CORRECT}| > n/2$. Since the adversary is oblivious, X_i is independent of whether process $p_i \in \text{CORRECT}$.

Since $\Pr(X_i) = 8c \log n/n$, we conclude that $\mathbb{E}(|C_0|) = 8c \log n$, and $\mathbb{E}(|C_0 \cap \text{CORRECT}|) \geq 4c \log n$. Thus, by a straightforward Chernoff bound, $\Pr(|C_0| \geq 16c \log n) \leq 1/n^c$, and $\Pr(|C_0 \cap \text{CORRECT}| \leq c \log n) \leq 1/n^c$. \square

Next, we show that the CHOOSEC protocol satisfies probabilistic uniformity, i.e., that, every pair of coordinators has approximately the same list of coordinators, with high probability. This follows by a straightforward birthday-paradox style analysis.

LEMMA 3.2. (PROBABILISTIC UNIFORMITY) *With probability at least $1 - 1/n^c$, there exists a subset S such that: (i) every process $p_i \in S$ is a coordinator ($isC_i = \text{TRUE}$); (ii) every non-failed coordinator at the end of the CHOOSEC protocol is a member of S ; (iii) for each non-failed coordinator $p_i \in S$, $coords_i \subseteq S$; (iv) for each non-failed coordinator $p_i \in S$, for each*

process $p_j \in S \setminus \text{coords}_i$, process p_j fails by the end of the CHOOSEC protocol.

Proof. Let S be the set of processes that set $\text{isC} = \text{TRUE}$ in Round 0. It follows that every process $p_i \in S$ is a coordinator, and that every non-failed coordinator is a member of S . Moreover, it is easy to see that for each coordinator $p_i \in S$, $\text{coords}_i \subseteq S$, as p_i only adds to coords_i processes that send messages to intermediaries in Round 1, i.e., p_i only adds other coordinators to coords_i . It remains to show the final property.

Fix some non-failed coordinators p_i and p_j . In Round 1, both p_i and p_j send messages to $\Theta(\sqrt{n} \log n)$ intermediaries. Let I be the set of intermediaries chosen by p_i in Round 1. The probability that p_j does not choose an intermediary in the set I is at most:

$$\begin{aligned} \left(1 - \frac{|I|}{n}\right)^{2c\sqrt{n} \log n} &\leq \left(1 - \frac{2c\sqrt{n} \log n}{n}\right)^{2c\sqrt{n} \log n} \\ &\leq \left(\frac{1}{2}\right)^{4c^2 \log^2 n} \\ &\leq \left(\frac{1}{n}\right)^{c+2} \end{aligned}$$

This implies that $p_j \in \text{coords}_i$ with probability at least $1 - 1/n^{c+2}$. Taking a union bound over all $\binom{n}{2}$ pairs of p_i and p_j , the probability that there exists some non-failed p_i and p_j such that $p_j \notin \text{coords}_i$ by the end of Round 2 is no greater than $1/n^c$. From this we conclude that, with high probability, for every coordinator p_i , for every $p_j \in S \setminus \text{coords}_i$, p_j fails by the end of the CHOOSEC protocol. \square

Finally, we conclude that the CHOOSEC protocol satisfies the requisite properties:

LEMMA 3.3. *The CHOOSEC protocol guarantees self-inclusion^{CC}, probabilistic uniformity^{CC}, coordinator-set size^{CC}, and termination^{CC}.*

Proof. The *self-inclusion* property follows immediately from the protocol, as every coordinator p_i adds itself to coords_i in Round 2. The *probabilistic uniformity* property follows immediately from Lemma 3.2. The *coordinator-set size* property follows from Lemma 3.1. Termination is immediate by inspection. \square

We bound the communication complexity of CHOOSEC by simply summing the message costs:

LEMMA 3.4. *With probability at least $1 - 1/n^c$, the CHOOSEC protocol has communication complexity $O(\sqrt{n} \log^4 n)$.*

Proof. By Lemma 3.1, there are at most $16c \log n$ coordinators with probability at least $1 - 1/n^c$. In this case: In Round 1, there are at most $16c \log n$ coordinators, each of which sends $2c\sqrt{n} \log n$ messages of size $\log n$ bits. In Round 2, there are at most $32c^2 \sqrt{n} \log^2 n$ responses that result from the Round 1 messages, and each response contains at most $16c \log n$ identifiers of size $\log n$ bits. Summing the bit complexities leads to the desired bound. \square

3.2 Limited Universe Consensus. The goal of the second sub-protocol, LUCONSENSUS, is to achieve agreement among the coordinators. The protocol itself is a simple implementation of deterministic consensus, as described in [23]. The only difference here is that the protocol is executed only on the coordinators, and with some (small) probability, the coordinators have different views of the world.

Each process p_i begins with three inputs: (1) an initial value v_i , (2) a boolean flag isC_i indicating whether process p_i is a coordinator, and (3) a list coords_i of other coordinators. We assume that the isC indicator and the coordinator lists coords satisfy self-inclusion, probabilistic uniformity, and coordinator-set size, as per the CHOOSEC protocol. Each process with $\text{isC} = \text{TRUE}$ returns one output: a value v^o . The sub-protocol guarantees the following properties at the end of the execution:

- *Probabilistic Agreement^{LUC}*: With high probability, every process that outputs a value chooses the same value.
- *Validity^{LUC}*: If process p_i outputs value v_i^o , then there is some process p_j with initial value $v_j = v_i^o$.
- *Termination^{LUC}*: The protocol terminates in $O(\log n)$ rounds. Every non-failed process with input $\text{isC} = \text{TRUE}$ produces an output.

Note that unlike the classical requirements of consensus, the LUCONSENSUS protocol only ensures that agreement is reached *with high probability*; with some small probability, the coordinators may disagree. This simplifies the LUCONSENSUS sub-protocol, compensating for the fact that with some small probability, the set of coordinators provided as part of the input may itself be inconsistent.

Sub-protocol description. Each process p_i maintains an estimate e_i . Initially, $e_i = v_i$. The LUCONSENSUS sub-protocol consists of the following, repeated for $\Theta(\log n)$ rounds, specifically for at least $16c \log n + 1$ rounds:

- Each coordinator p_i with $\text{isC}_i = \text{TRUE}$ sends its estimate e_i to every process in coords_i .

- Each coordinator p_i with $isC = \text{TRUE}$ updates its estimate with the minimum estimate received in that round.

After $\Theta(\log n)$ rounds, each coordinator p_i with $isC = \text{TRUE}$ outputs its estimate e_i .

Analysis. The LUCONSENSUS is, essentially, an implementation of the classical FLOODSET algorithm (as presented in [23]) on a limited universe. The key component of the proof is showing that the *probabilistic uniformity* property is sufficient to ensure that the coordinators correctly execute consensus.

LEMMA 3.5. *The LUCONSENSUS protocol satisfies probabilistic agreement^{LUC}, validity^{LUC}, and termination^{LUC}, assuming the inputs isC and $coords$ satisfy self-inclusion^{CC}, probabilistic uniformity^{CC}, and coordinator-set size^{CC}.*

Proof. Validity^{LUC} and termination^{LUC} follow immediately, by inspection. In order to see that agreement^{LUC} holds, recall that *probabilistic uniformity^{CC}* and *coordinator set size^{CC}* hold with respect to the inputs (isC and $coords$). In particular, with high probability, there exists a subset S of size $\Theta(\log n)$ such that: (i) every process $p_i \in S$ is a coordinator; (ii) every *non-failed* coordinator is a member of S ; (iii) for each non-failed coordinator $p_i \in S$, $coords_i \subseteq S$; (iv) for each non-failed coordinator $p_i \in S$, for each process $p_j \in S \setminus coords_i$, process p_j fails by the end of the CHOOSEC protocol.

From this we conclude, by the pigeon-hole principle, that there is some round r during the $\Theta(\log n)$ rounds of the LUCONSENSUS protocol in which no process in the subset S fails; fix such a round r . From the set of processes in S that do not fail by the end of round r , choose p_i to be a process with the minimum estimate. In round r , process p_i sends its estimate to every process in $coords_i$.

Let p_j be some other coordinator that does not fail by the end of round r . By probabilistic uniformity, we know that $p_j \in coords_i$, with high probability; otherwise, $p_j \in S \setminus coords_i$, implying that it fails by the end of the CHOOSEC protocol, i.e., prior to round r . Thus p_j receives a message from p_i containing e_i ; since e_i is the minimum estimate among non-failed coordinators, p_j adopts estimate e_i . From this we conclude that at the end of round r , every non-failed coordinator has adopted estimate e_i . This immediately implies the agreement property, as desired. \square

The communication complexity can be calculated by summing the various message costs:

LEMMA 3.6. *With high probability, the communication complexity of LUCONSENSUS is $O(\log^3 n)$, assuming the*

inputs isC and $coords$ satisfy self-inclusion^{CC}, probabilistic uniformity^{CC}, and coordinator-set size^{CC}.

Proof. As per the coordinator-set size property, with high probability there are at most $O(\log n)$ coordinators. In each round, each coordinator sends a constant-sized message to every other coordinator, and this continues for $O(\log n)$ rounds, leading to the desired bound. \square

3.3 Coordinator Data Dissemination. The goal of the third sub-protocol, DISSEMINATE, is to efficiently disseminate data from the set of coordinators to all the other processes. Each process p_i begins with three inputs: (1) an initial value v_i , (2) a boolean flag isC_i indicating whether process p_i is a coordinator, and (3) a list $coords_i$ of other coordinators, where the initial values v_i satisfy probabilistic agreement, and isC_i and $coords_i$ satisfy self-inclusion, probabilistic uniformity, and coordinator-set size. Each process p_i outputs a set of values V_i ; each coordinator p_i outputs a flag ds_i indicating whether its initial value was successfully disseminated. The sub-protocol guarantees the following properties at the end of the execution:

- *Dissemination^D*: For every non-failed coordinator p_i , the initial value v_i is sent to every process, i.e., for every non-failed p_j , $v_i \in V_j$.
- *Validity^D*: If, for some process p_i , there is some value $v \in V_i$, then some coordinator p_j initiated DISSEMINATE with value v_j , i.e., there exists a p_j where $isC_j = \text{TRUE}$ and $v_j = v$.
- *Consistency^D*: If, for some pair of processes p_i, p_j , the flag $ds_i = \text{TRUE}$ and $ds_j = \text{TRUE}$, then initial values $v_i = v_j$.
- *Termination^D*: The protocol terminates in $O(\log^* n)$ rounds. With high probability, every correct coordinator p_i returns $ds_i = \text{TRUE}$.

Sub-protocol description. Partition the processes in Π into $\log n \log^* n$ groups $G_1, G_2, \dots, G_{\log n \log^* n}$ each of size $n/(\log n \log^* n)$. Each process p_i maintains a list of unnotified groups L_i and a count c_i of the number of notified processes. Initially, L_i contains every group, and $c_i = 0$. The protocol proceeds in $\Theta(\log^* n)$ triples of rounds; the constant factor follows from the analysis. Each triple of rounds proceeds as follows:

Round 1. Each coordinator p_i chooses a group g at random from the list L_i , and sends its value v_i to every process in group g .

Round 2. Each process p_j that received exactly one value v in Round 1, and that did not previously

receive a different value $v' \neq v$ in a previous round, adds v to the set V_j and sends a response. (Note that if p_j received the same value v in a previous round, then it still sends a response.) Each coordinator, on receiving the responses from processes in group g , counts the number n_g of responses received.

Round 3. Each coordinator p_i sends $\langle g, n_g, v_i \rangle$ to every other coordinator in coords_i . When a coordinator p_j receives a message containing $\langle g', n'_g, v' \rangle$, it proceeds as follows: if $g' \in L_j$ and $v' = v_j$, then coordinator p_j removes g' from L_j and adds n'_g to c_j .

These three rounds are repeated $\Theta(\log^* n)$ times. At this point, if, for some coordinator p_i , the list L_i is not empty, then p_i proceeds as follows:

- Coordinator p_i sends the value v_i directly to every process in Π .
- Every process that receives such a message sends a response according to the same conditions as before, i.e., only if it has not received a different value in an earlier round.
- The coordinator p_i sets the count c_i to the number of responses received (overwriting any earlier value of c_i).

At this point, every coordinator is ensured that its initial value has been disseminated, either directly or indirectly. If count $c_i > n/2$, then the coordinator p_i returns $ds_i = \text{TRUE}$; if count $c_i \leq n/2$, then the coordinator p_i returns $ds_i = \text{FALSE}$. Every process $p_j \in \Pi$ returns V_j .

Analysis. We now show that the DISSEMINATE protocol satisfies the requisite properties. The *disseminate*^D property follows from the simple fact that a process p_i only removes a group g from its list if it sends a message to every process in g , or if another coordinator with the same initial value sends a message to every process in g :

LEMMA 3.7. (DISSEMINATION) *For every non-failed coordinator p_i , the initial value v_i is sent to every process, i.e., for every non-failed p_j , $v_i \in V_j$.*

Proof. We first argue that if a group g is removed from the list L_i by a process p_i , then every process in g has been sent the value v_i . There are two reasons a group may be removed from L_i . First, it might be the case that p_i directly sends value v_i to every process in g . Second, it might be the case that p_i receives a message from some other process p_j indicating that p_j has sent

message $v_j = v_i$ to every process in g . In both cases, the removal of g clearly implies that every process in g has been sent the value v_i . At the end of the protocol, either L_i is empty, in which case v_i has been sent to every process in Π , or process p_i proceeds to send v_i directly to every process. \square

We next argue that the DISSEMINATE protocol satisfies the *consistency*^D property:

LEMMA 3.8. (CONSISTENCY) *If, for some pair of processes p_i, p_j , the flag $ds_i = \text{TRUE}$ and $ds_j = \text{TRUE}$, then initial values $v_i = v_j$.*

Proof. We first argue that for every coordinator p_i , the count c_i is a lower bound on the number of processes that received value v_i before receiving any other values. If p_i broadcasts a message directly to every process (after $\Theta(\log n)$ triples of rounds), then the count c_i is set to exactly the number of responses received; every such response indicates a process that received no value other than v_i in a prior round. Consider the case where p_i does not broadcast a message directly to every process. Observe that for each group g , a process p_i adds count n_g to c_i only once, i.e., when g is removed from L_i ; and the count n_g reflects the number of responses received by some coordinator p_k that sent $v_k = v_i$ to every process in the group g . Thus again we conclude that there are at least c_i processes that received no value other than v_i prior to receiving value v_i .

Now, we conclude the proof. Fix two processes p_i and p_j such that $ds_i = ds_j = \text{TRUE}$ when the protocol completes. This implies that $c_i > n/2$ and $c_j > n/2$. Thus there is some process p_ℓ that is included in both counts, i.e., that received a message containing v_i prior to (and not concurrently with) receiving any other value, and that also received a message containing v_j prior to (and not concurrently with) receiving any other value; this implies that $v_i = v_j$, as desired. \square

We conclude that the DISSEMINATE protocol satisfies the requisite properties:

LEMMA 3.9. *The DISSEMINATE protocol satisfies dissemination^D, validity^D, consistency^D, and termination^D, assuming the inputs isC and coords satisfy self-inclusion^{CC}, probabilistic uniformity^{CC}, and coordinator-set size^{CC}, and assuming the input initial values satisfy probabilistic agreement^{LUC}.*

Proof. The dissemination and consistency properties follow from Lemma 3.7 and Lemma 3.8, respectively. Validity follows from simple inspection, as a process only returns values received directly from coordinators.

Similarly, it is immediately clear that the protocol terminates in $O(\log^* n)$ rounds. Finally, since the initial values agree, with high probability, then every process responds to every dissemination message from every coordinator; since a majority of processes are correct, the successful dissemination implies that every non-failed coordinator p_i outputs $ds_i = \text{TRUE}$. \square

We now show that the DISSEMINATE protocol has communication complexity $O(n)$, with high probability. The key claim is as follows: if the set of coordinators contains $\Theta(\log n)$ correct processes, then after $\Theta(\log^* n)$ rounds, every group has been sent the initial value by some coordinator, without resorting to the final round in which coordinators send their value directly to everyone. In this case, the total communication incurred is $\Theta(\log^* n)$ triples of rounds in which $\Theta(\log n)$ processes send (and receive as responses) $\Theta(n/(\log n \log^* n))$ bits each, along with a small amount of inter-coordinator communication.

We say that a group is *notified* (with respect to S) if it has been selected, and its processes have been sent a message, by some coordinator in S . Otherwise, it is *unnotified*. We model the process of notifying groups as a classic balls-and-bins game: each of the $\log n \log^* n$ groups is a bin, and in each round, each coordinator throws one “ball” into one “bin,” notifying a particular group. After each round, every “bin” that has received at least one “ball” is removed from the game (via coordination among the coordinators). We refer to this as the process of *clearing bins*.

We first establish that, for a given group of coordinators S of size at least $\log n$, within $O(\log^* n)$ rounds, there are at most $2 \log n$ unnotified groups. This follows from the observation that in each round, each coordinator has a probability of at least $1/2$ of selecting a previously unnotified group, resulting in a reduction of the number of unnotified groups by $\Theta(\log n)$:

LEMMA 3.10. *If subset S contains at least $\log n$ correct coordinators, then within $O(\log^* n)$ rounds there are at most $2 \log n$ groups that are unnotified, with high probability.*

Proof. Assume for the sake of contradiction that for $c \log^* n$ triples of rounds, there are at least $2 \log n$ unnotified groups. We focus on the behavior of exactly $\log n$ coordinators (ignoring the notifications by additional coordinators). Observe that in each such triple of rounds, each of the $\log n$ coordinators in S notifies a new group with probability at least $\frac{\log n}{2 \log n} = \frac{1}{2}$, independent of the choices made by the other coordinators. For $\log n$ coordinators, over $c \log^* n$ rounds, this implies that in

expectation there are at least $c \log n \log^* n$ groups notified. Thus by a Chernoff bound³, we observe that the probability of notifying less than $(c/4) \log n \log^* n$ groups is no greater than $e^{-c \log n \log^* n/4} \leq (1/n)^{c/4}$. When $c \geq 4$, this is a contradiction (as there are initially $\log n \log^* n$ groups). \square

We now examine the bin-clearing process when there are at least $\log n$ coordinators and at most $2 \log n$ remaining groups. This random process has been previously studied in [9]⁴:

LEMMA 3.11. ([9], THEOREM 2) *The process of clearing b bins with ℓ balls, for $\ell \geq b/2$, terminates within $\log^*(b \log \ell / \ell) + O(1)$ rounds with probability at least $1 - 2e^{-\ell^{1/5}}$.*

From this, we derive the straightforward corollary, applying this to our situation:

COROLLARY 3.1. *If, for some subset S , there are at most $2 \log n$ unnotified groups, and at least $\log n$ correct coordinators, then within $O(\log^* n)$ rounds, every group has been notified, with high probability.*

Proof. Applying Lemma 3.11, where $b \leq 2 \log n$ and $\ell = \log n$, we observe that after $\log^*(2 \log \log n) \leq \log^* n$ rounds, with probability at least $1 - (1/n)^{(1/5) \ln 2}$, every group has been notified. Thus after some $O(c \log^* n)$ rounds, every group has been notified with probability polynomially small in n . \square

Putting together Lemma 3.10 and Corollary 3.1, we conclude the following:

LEMMA 3.12. *The communication complexity of DISSEMINATE is $O(n)$, with high probability, assuming the inputs isC and coords satisfy self-inclusion^{CC}, probabilistic uniformity^{CC}, and coordinator-set size^{CC}, and assuming the input initial values satisfy probabilistic agreement^{LUC}.*

Proof. With high probability, there is a subset S that satisfies probabilistic uniformity and coordinator set size, by assumption; and with high probability, the initial values are all the same, by assumption. By

³Formally, the events are not independent, however due to the property that notification of a new group happens with probability at least $1/2$ independently of other choices, we can stochastically estimate the number of notified groups by the sum of independent variables with expected value $1/2$.

⁴In fact, Lemma 3.11 is a slight generalization of the theorem in [9], as we assume $\ell \geq b/2$, rather than $\ell > b$. Adding only a constant number of rounds in the beginning of the process ensures that the number of bins drops down to at most $b/2$.

Corollary 3.1, with high probability, within $O(\log^* n)$ rounds, every group has been notified by a non-failed coordinator in S . Consider the case where these high probability events occur.

We now argue that every non-failed coordinator p_i removes every group from its list L . Let g be a group notified by a non-failed coordinator $p_j \in S$, and let p_i be another non-failed coordinator. If $p_i \notin \text{coords}_j$, then $p_i \in S \setminus \text{coords}_j$, and hence we conclude that p_i has failed. Thus, since p_i is non-failed, we know that $p_i \in \text{coords}_j$, and hence receives the update from p_j that g was notified. Since all coordinators agree on the same value, we know that $v_j = v_i$, and hence p_i removes g from L_i . By the end of the protocol, every coordinator has an empty group list, and hence no coordinator sends its value directly to every process in Π .

The communication complexity can be divided into three parts: sending messages to notify groups, sending responses, and sending messages among coordinators. In each round, each coordinator sends $n/\log n \log^* n$ messages (i.e., one message to each process in a group), and each message is of size $O(1)$, containing only the value to be disseminated. Each such message leads to a single response of size $O(1)$. There are at most $O(\log^* n)$ rounds, and at most $O(\log n)$ coordinators, and thus the total bit complexity of such messages is $O(n)$. Communication among coordinators consists of $O(\log^2 n)$ messages in each round, each of size $\leq \log(\log n \log^* n) + 1 + \log n$ —i.e., containing the name of the most recently notified group, the value to be disseminated, and a count of processes. Thus, the total communication complexity is $O(n + \log^3 n)$. \square

3.4 Complete Consensus Protocol. We now present the complete protocol. Each process p_i begins with initial value v_i , and maintains an estimate e_i ; initially $e_i = v_i$. Process p_i executes the following steps:

Communication-Optimal Consensus Protocol

1. CHOOSEC(\cdot) $_i \rightarrow \langle \text{is}C_i, \text{coords}_i \rangle$
2. LUCONSENSUS($v_i, \text{is}C_i, \text{coords}_i$) $_i \rightarrow v$
 - Set $e_i = v$.
3. DISSEMINATE($e_i, \text{is}C_i, \text{coords}_i$) $_i \rightarrow \langle V_i, \text{ds}_i \rangle$
 - If $\text{is}C_i = \text{TRUE}$ and $\text{ds}_i = \text{FALSE}$, then set $\text{is}C_i = \text{FALSE}$.
4. DISSEMINATE($e_i, \text{is}C_i, \text{coords}_i$) $_i \rightarrow \langle V_i, \text{ds}_i \rangle$
 - If $\text{is}C_i = \text{TRUE}$ and $\text{ds}_i = \text{FALSE}$, then set $\text{is}C_i = \text{FALSE}$.

- If $\text{is}C_i = \text{FALSE}$ and V_i is not empty, then set e_i to the unique value in V . (We will show that there is always at most one value in the set V .)

5. DISSEMINATE($e_i, \text{is}C_i, \text{coords}_i$) $_i \rightarrow \langle V_i, \text{ds}_i \rangle$
 - If there is a value $v \in V_i$ and $v = e_i$, then decide v .
6. Each undecided process sends a message containing a “fallback request” to every other process requesting that they begin the fallback protocol.
7. Every process that has not decided, or that receives a “fallback request” in Step 6 executes a classical *consensus* protocol (e.g., the FLOODSET Protocol [23]), using value e_i as its initial value. Every process that has not yet decided adopts the value returned and decides.

We now argue that the resulting protocol is correct.

THEOREM 3.1. (CORRECTNESS) *The communication-optimal consensus protocol satisfies agreement, validity, and termination.*

Proof. We begin by showing that the protocol satisfies validity. Specifically, validity follows from the *validity*^{LUC} property of LUCONSENSUS and the *validity*^D property of the DISSEMINATE protocol: every value e_i that is eventually decided is either delivered by the DISSEMINATE protocol or output by the LUCONSENSUS protocol; every value distributed by the DISSEMINATE protocol was previously output by the LUCONSENSUS protocol; and every value output by the LUCONSENSUS protocol was previously the initial value of some process.

It is also easy to see that the protocol satisfies termination, as every sub-protocol terminates. More specifically, termination follows immediately from the *termination*^{CC} property of CHOOSEC, the *termination*^{LUC} property of LUCONSENSUS, the *termination*^D property of the DISSEMINATE protocol, and the *termination* property of the fallback consensus protocol.

We now consider the agreement property. We first argue that at the end of Step 3, every non-failed coordinator has the same value. Assume that, at the end of Step 3, there exist two non-failed coordinators p_i and p_j such that: $\text{is}C_i = \text{TRUE}$ and $\text{is}C_j = \text{TRUE}$. From this we conclude that both p_i and p_j completed the DISSEMINATE protocol successfully in Step 3, i.e., $\text{ds}_i = \text{TRUE}$ and $\text{ds}_j = \text{TRUE}$. By the *consistency* property of DISSEMINATE, we conclude that $e_i = e_j$. Thus, every

non-failed coordinator has the same candidate value at the end of Step 3.

Assume for the sake of contradiction that there are two distinct values v and w decided by the end of the protocol. By the *agreement* property of the fallback consensus protocol, we know that only one value is decided in Step 7. Thus either v or w is decided in Step 5. Assume, without loss of generality, that value v is decided in Step 5 by some process.

From this we conclude that the DISSEMINATE protocol in Step 5 returned value v in the set V_k for some process p_k , and hence (by *validity* ^{\mathcal{P}}), some coordinator p_j initiated the DISSEMINATE protocol in Step 5 with $e_j = v$ and $isC_j = \text{TRUE}$. Thus, in Step 4, process p_j disseminated value v , i.e., the *dissemination* property implies that every process received value v in Step 4. As every coordinator has the same candidate value at the end of Step 3, we conclude that v is the only value received in Step 4, and hence every non-failed process p_i adopts v , setting $e_i = v$.

This leads us to two conclusions. First, in Step 5, since every non-failed process p_i has $e_i = v$, no process can decide w in Step 5. Second, every non-failed process p_i still has $e_i = v$ at the beginning of Step 7, and hence by *validity* of the fallback consensus protocol, every non-failed process outputs v in Step 7. Thus no process decides w , which is a contradiction. \square

Finally, we conclude that the protocol achieves good bit complexity:

THEOREM 3.2. (EFFICIENCY) *The communication-optimal consensus protocol has $O(n)$ communication complexity and $O(\log n)$ round complexity, with high probability.*

Proof. With high probability, the set of coordinators (and the set of *correct* coordinators) chosen by CHOOSEC is of size $\Theta(\log n)$; that is, in every round after CHOOSEC completes, there are $\Theta(\log n)$ non-failed coordinators. In addition, the set of coordinators satisfies probabilistic uniformity, as per the properties of the CHOOSEC sub-protocol. Moreover, with high probability, every process agrees on the same estimate v in Step 2, as per the properties of the LUCONSENSUS sub-protocol. And, with high probability, every correct coordinator returns $ds = \text{TRUE}$ in Step 3, as per the properties of the DISSEMINATE protocol. Every process receives and adopts value v in Step 4, and decides v in Step 5, as per the properties of the DISSEMINATE protocol. From this we conclude that no process proceeds to the fallback phase, skipping Steps 6–7. Thus, the total bit complexity is the cost of CHOOSEC (Lemma 3.4), plus the cost of LUCONSENSUS (Lemma 3.6), plus the

cost of three invocations of DISSEMINATE (Lemma 3.12), which totals $O(n)$. \square

4 Partially Synchronous Systems

While networks are typically *synchronous*, delivering messages in a timely fashion, real protocols must cope with occasional *asynchrony*, i.e., unpredictable message delays. Thus it is considered good computing practice to plan for the worst and hope for the best. In the context of distributed computing, this translates into devising algorithms that, on the one hand tolerate asynchrony, while on the other hand, operate efficiently when the network is synchronous.

We refer to systems that are typically synchronous, but occasionally asynchronous, as *partially synchronous systems*. In this section, we show how to modify the algorithm described in Section 3 so that it can be safely deployed in a partially synchronous network, maintaining its high level of efficiency whenever the network is, in fact, synchronous.

4.1 Model. In this section we formally define a *partially synchronous system*, following the basic model in [14]. The system is parameterized by three variables: n , the number of processes, δ a synchronous bound on clock skew, and d a synchronous bound on message delay. We assume that n , δ and d are known *a priori*.

As before, we consider a set Π of crash-prone processes p_1, \dots, p_n , a majority of which are correct. Each process has its own local clock that proceeds at an arbitrary rate. As before, processes communicate directly with each other; however, we *do not* assume that communication proceeds in rounds. Instead, a message may be arbitrarily delayed. Every message sent by a correct process to some other correct process is eventually delivered.

We say that an execution is *synchronous* when the following conditions hold: (i) the clock skew of every process is bounded by δ , i.e., the ratio of the rates of two processes' clocks is at most δ ; and (ii) every message is delivered within d time. Otherwise, we say that an execution is *asynchronous*.

4.2 Protocol Modifications. We now describe how to modify the communication-optimal consensus protocol presented in Section 3 in order to operate safely in a partially synchronous system, while remaining efficient in synchronous executions.

First, each process simulates synchronous rounds based on message delay d and clock skew δ . Observe that in a synchronous execution, at time τ the clock at every process i is in the range $[(1 - \delta)\tau, (1 + \delta)\tau]$.

Let $\rho = (1 + \delta)/(1 - \delta)$. The first simulated round r_1 is defined, for process p_i , to end at time $d/(1-\delta)$ according to the local clock at process p_i . Simulated round r is defined for process p_i to end at time: $\frac{d}{1-\delta} \sum_{j=0}^{r-1} \rho^j$ according to the local clock of process p_i .

Since the clocks of two processes p_i and p_j may advance at different rates, there is no guarantee that both processes begin and end their rounds at the same time. However, in a synchronous execution, the clock skew is bounded by δ , and hence every message sent by p_i to p_j at the beginning of round r , according to the local clock of p_i , is received by the end of round r according to the local clock of process p_j . In an asynchronous execution, however, the simulation of synchronous rounds may, of course, fail completely.

Second, each process p_i that sends a fallback request message in Step 6 of the protocol acts as follows: if p_i received a value in Step 4, then it attaches its current estimate e_i to the fallback message. When a process p_j receives a fallback message, it immediately aborts the ongoing protocol, jumping immediately to fallback Step 6. (Of particular note, p_j sends no further responses as part of the DISSEMINATE protocol.)

Third, a process p_i does not begin the fallback consensus protocol in Step 7 until it has received fallback request messages from at least a majority of the processes. When process p_i receives a fallback request message from some process p_j with estimate e_j , it adopts that estimate, setting $e_i = e_j$.

Finally, when choosing a fallback consensus protocol for Step 7, we focus only on protocols that operate correctly in asynchronous networks. For example, we might use the asynchronous consensus protocol in [16], which guarantees expected sub-quadratic message complexity in asynchronous executions.

4.3 Analysis. In this section, we first observe that the modified communication-optimal consensus protocol still guarantees good performance in synchronous executions:

THEOREM 4.1. *In a synchronous execution, the modified communication-optimal consensus protocol has $O(n)$ communication complexity and $O(\log n)$ simulated round complexity, with high probability.*

Proof. When the execution is synchronous, the simulation of synchronous rounds is correct, delivering every message from a non-failed process in the round in which it was sent. Thus, the protocol proceeds exactly as in a synchronous execution and the communication/time complexity are exactly as in Theorem 3.2. \square

It remains to argue that the protocol is correct, even in asynchronous executions:

THEOREM 4.2. *The modified communication-optimal consensus protocol guarantees agreement, validity, and termination.*

Proof. As before, validity follows immediately by inspection. Termination is also straightforward: any correct process that does not terminate eventually sends out a fallback request message; eventually every non-failed process receives this message and sends a fallback request message; since a majority of processes are correct, eventually every non-failed process receives enough fallback request messages and begins the asynchronous fallback consensus protocol; eventually, the fallback consensus protocol terminates, allowing every correct process to decide.

Agreement follows from the observation that the safety of the DISSEMINATE protocol does not depend on synchrony. As before, the *dissemination*^D property follows from the fact that a non-failed coordinator p_i always sends its initial value v_i directly to every process that has not already been sent the value v_i . The *consistency*^D property follows from the fact that a coordinator p_i returns $ds_i = \text{TRUE}$ only if a majority of processes receive value v_i prior to any other value. The *validity*^D and *termination*^D properties are similarly unaffected by asynchrony (notably, the protocol terminates when all the rounds of the protocol have been simulated, according to the local clock).

Thus, as before, every non-failed coordinator (that still has $isC = \text{TRUE}$) has the same estimate by the end of Step 3. This ensures that at most one value is decided in Step 5. The agreement property of the fallback consensus protocol ensures that at most one value is decided in Step 7. It remains to consider the case where some value v is decided in Step 5. This implies that a majority of processes received (and responded, confirming reception of) that estimate in Step 4. Thus, any process that begins the fallback protocol receives that estimate from at least one process during the fallback request phase. The validity of the fallback consensus protocol ensures that v is the only possible decision in Step 7. \square

5 Gossip

In this section, we provide a brief overview of how to adapt the techniques presented thus far to solve the problem of gossip.

Assume each process p_i begins with a rumor r_i . The goal is for every process to learn the rumor of every other correct process. The same techniques described in

Section 3 yield a synchronous (or partially synchronous) protocol that can solve this problem of gossip. The key observation is that the DISSEMINATE routine can be used just as well to collect data as to distribute it.

Specifically, we modify the DISSEMINATE routine as follows. We refer to the modified protocol as COLLECT.

- First, when executed at process p_i , it takes three parameters: (i) r_i , a rumor to collect, (ii) isC_i , a flag indicating whether p_i is a coordinator, and (iii) $coords_i$, the set of coordinators. It returns to each coordinator p_i a set R_i of rumors and a flag ds_i ; it does not return anything to non-coordinators.
- Second, every process attaches its rumor r_i to the response sent to the coordinators during Round 2 in each triple of rounds. Each coordinator attaches the set of newly learned rumors to every message sent to the other coordinators during Round 3 in each triple of rounds. (Note that if the coordinators are attempting to aggregate data as it is collected, they may send some aggregate or compressed version of the newly discovered rumors.)

The COLLECT protocol has essentially the same properties (of validity, consistency, and termination) as the DISSEMINATE protocol, with the exception of the *dissemination*^D property which is now replaced by the *collection*^C property: For every non-failed coordinator p_i , for every non-failed process p_j , the rumor r_j is returned to p_i , i.e., $r_j \in R_i$.

The gossip protocol, then, proceeds as follows:

Communication-Efficient Gossip

1. CHOOSEC() $\rightarrow \langle isC_i, coords_i \rangle$
2. COLLECT($r_i, isC_i, coords_i$) $\rightarrow \langle R_i, ds_i \rangle$
3. DISSEMINATE($R_i, isC_i, coords_i$) $\rightarrow \langle V_i, ds_i \rangle$
4. DISSEMINATE(DONE, $isC_i, coords_i$) $\rightarrow \langle V_i, ds_i \rangle$
5. If process p_i has not received the flag DONE, then it sends a fallback request to every other process. Every process, on receiving such a request, sends its rumor in response.

The resulting protocol ensures that every rumor originating at a correct process is distributed to every non-failed process: if there are no non-failed coordinators that complete Step 4, then no process receives the DONE message and every process sends its rumor to all other process during the fallback Step 5; if there is even one non-failed coordinator, then by the collection property

it retrieves all the rumors in Step 2, and by the dissemination property it distributes all the rumors in Step 3.

It is also easy to see that the resulting protocol terminates in $O(\log^* n)$ rounds, as each individual step completes in at most $O(\log^* n)$ rounds.

Finally, with high probability, the gossip protocol uses no more than $O(n)$ messages. This follows from the probabilistic uniformity and coordinator-set size properties of the CHOOSEC protocol, along with the message-complexity analysis of the COLLECT/DISSEMINATE protocols.

While the protocol sends at most $O(n)$ message, with high probability, the communication complexity depends on the size of the rumors and their capacity to be efficiently aggregated. (This is, of course, inherent to all gossip protocols.) Even so, there are a wide variety of examples in which the data being collected can be efficiently aggregated (or summarized). For example, a common problem in distributed computing is verifying whether there is one or more than one non-failed leader in a system; such a problem can be solved using this gossip protocol with only $O(n)$ bits of communication.

6 Conclusion

In this paper, we have developed a new consensus algorithm that achieves optimal communication complexity and almost optimal time complexity. We have also shown how to modify the algorithm so that it can tolerate asynchronous executions, while maintaining good performance if the network is synchronous. The techniques we have used are quite general, and can be used to solve a variety of other problems. As an example, we have shown how to use this technique of *universe reduction* to efficiently gossip rumors.

One of the important open questions regards the property of *locality*, i.e., where no process sends more than $O(\log^{O(1)} n)$ bits. (In the communication-optimal consensus protocol in this paper, each of the coordinators may send $\Theta(n/\log n)$ messages.) We conjecture that the same techniques in this paper can be used to achieve locality; however straightforward modifications lead to larger communication and time complexity. In fact, there may well be an inherent trade-off between locality and communication/time complexity. The natural question, then, is what is the optimal communication complexity for a *local* algorithm?

The other major open question is resolving the optimal communication complexity for *deterministic* consensus algorithms with linear time complexity. (Recall that [15] achieves optimal $O(n)$ message complexity, but requires super-linear round complexity.) We conjecture that it is impossible to achieve both $O(n)$ communica-

tion complexity and $O(n)$ time complexity with a deterministic algorithm.

Acknowledgments

We would like to thank Valerie King for several helpful conversations, and especially her insights regarding the technique of *universe reduction*. We would also like to thank Dan Alistarh for many discussions regarding how to reduce the message complexity of distributed algorithms.

References

- [1] S. Amdur, S. Weber, and V. Hadzilacos. On the message complexity of binary agreement under crash failures. *Distributed Computing*, 5(4):175–186, 1992.
- [2] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.
- [3] H. Attiya and K. Censor. Lower bounds for randomized consensus under a weak adversary. In *Proceedings of the Twenty-Seventh Symposium on Principles of Distributed Computing (PODC)*, pages 315–324. ACM, 2008.
- [4] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. *J. of the ACM*, 55(5):1–26, 2008.
- [5] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine agreement in the full-information model in $o(\log n)$ rounds. In *Proceedings of the Thirty-Eighth Symposium on Theory of Computing (STOC)*, pages 179–186. ACM, 2006.
- [6] B. Chlebus and D. Kowalski. Gossiping to reach consensus. In *Proceedings of 14th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 220–229, 2002.
- [7] B. Chlebus and D. Kowalski. Robust gossiping with an application to consensus. *Journal of Computer and System Science*, 72(8):1262–1281, 2006.
- [8] B. Chlebus and D. Kowalski. Locally scalable randomized consensus for synchronous crash failures. In *Proceedings of 21st Symposium on Parallel Algorithms and Architectures (SPAA)*, 2009.
- [9] B. S. Chlebus and D. R. Kowalski. Randomization helps to perform independent tasks reliably. *Random Struct. Algorithms*, 24(1):11–41, 2004.
- [10] B. S. Chlebus, D. R. Kowalski, and M. Strojnowski. Fast scalable deterministic consensus for crash failures. In *Proceedings of the 28th Symposium on Principles of Distributed Computing (PODC)*, 2009.
- [11] B. Chor, M. Merritt, and D. B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *J. of the ACM*, 36(3):591–614, 1989.
- [12] D. Dolev and H. Strong. Requirements for agreement in a distributed system. Technical Report RJ 3418, IBM Research, San Jose, CA, Mar. 1982.
- [13] C. Dwork, J. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *SIAM Journal on Computing*, 27(5):1457–1491, 1998.
- [14] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.
- [15] Z. Galil, A. Mayer, and M. Yung. Resolving message complexity of byzantine agreement and beyond. In *Proceedings of the 36th Symposium on Foundations of Computer Science (FOCS)*, pages 724–733, 1995.
- [16] C. Georgiou, S. Gilbert, R. Guerraoui, and D. Kowalski. On the complexity of asynchronous gossip. In *Proceeding of the 27th Symposium on Principles of Distributed Computing (PODC)*, 2008.
- [17] S. Gilbert, R. Guerraoui, and D. Kowalski. On the message complexity of indulgent consensus. In *Proceedings of the the 21st International Symposium on Distributed Computing (DISC)*, 2007.
- [18] B. M. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. In *Proceedings of the Nineteenth Annual Symposium on Discrete Algorithms (SODA)*, pages 1038–1047, 2008.
- [19] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vekking. Randomized rumor spreading. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS)*, 2000.
- [20] V. King and J. Saia. Fast, scalable byzantine agreement in the full information model with a nonadaptive adversary. In *Proceedings of the 23rd International Symposium on Distributed Computing (DISC)*, 2009.
- [21] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual Symposium on Discrete Algorithms (SODA)*, pages 990–999, 2006.
- [22] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ToPLaS*, 4(3):382–401, 1982.
- [23] N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.
- [24] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic quorum systems. *Information and Computation*, 170(2):184–206, 2001.
- [25] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.