On the Complexity of Asynchronous Gossip

Chryssis Georgiou
Dept. of Computer Science,
University of Cyprus,
CY-1048 Nicosia, Cyprus.
chryssis@cs.ucy.ac.cy

Seth Gilbert
École Polytechnique
Fédérale de Lausanne,
1015 Lausanne, Switzerland.
seth.gilbert@epfl.ch

Rachid Guerraoui
École Polytechnique
Fédérale de Lausanne,
1015 Lausanne, Switzerland.
rachid.guerraoui@epfl.ch

Dariusz R. Kowalski
Dept. of Computer Science,
University of Liverpool,
Liverpool L69 3BX, UK.
D.Kowalski@liverpool.ac.uk

ABSTRACT

In this paper, we study the complexity of *gossip* in an asynchronous, message-passing fault-prone distributed system. In short, we show that an *adaptive* adversary can significantly hamper the spreading of a rumor, while an *oblivious* adversary cannot. This latter fact implies that there exist message-efficient asynchronous (randomized) *consensus* protocols, in the context of an oblivious adversary.

In more detail, we summarize our results as follows. If the adversary is adaptive, we show that a randomized *asynchronous* gossip algorithm cannot terminate in fewer than $O(f(d+\delta))$ time steps unless $\Omega(n+f^2)$ messages are exchanged, where n is the total number of processes, f is the number of tolerated crash failures, d is the maximum communication delay for the specific execution in question, and δ is the bound on relative process speeds in the specific execution. The lower bound result is to be contrasted with deterministic *synchronous* gossip algorithms that, even against an adaptive adversary, require only $O(\text{ polylog}(n))$ time steps and $O(n \text{ polylog}(n))$ messages.

In the case of an oblivious adversary, we present three different randomized, asynchronous algorithms that provide different trade-offs between time complexity and message complexity. The first algorithm is based on the epidemic paradigm, and completes in $O(\frac{n}{n-f}\log^2 n(d+\delta))$ time steps using $O(n\log^3 n(d+\delta))$ messages, with high probability. The second algorithm relies on more rapid dissemination of the rumors, yielding a constant-time (w.r.t. n) gossip protocol: for every constant $\varepsilon < 1$, and for $f \le n/2$, there is a variant with time complexity $O(\frac{1}{\varepsilon}(d+\delta))$ and message complexity $O(\frac{1}{\varepsilon}n^{1+\varepsilon}\log n(d+\delta))$. The third algorithm solves a weaker version of the gossip problem in which each process receives at least a majority of the rumors. This algorithm achieves constant $O(d+\delta)$ time complexity and message complexity $O(n^{7/4}\log^2 n)$.

As an application of these message-efficient gossip protocols, we present three randomized consensus protocols. Our consensus algorithms derive from combining each of our gossip protocols with the Canetti-Rabin framework, resulting in message-efficient consensus algorithms. The resulting protocols have time and message-complexity asymptotically equal to our gossip protocols. We particularly highlight the third consensus protocol, a result that is interesting in its own right: the first asynchronous randomized consensus algorithm with strictly subquadradic message-complexity, i.e., $O(n^{7/4}\log^2 n)$.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: [fault tolerance]; G.3 [**Probability and Statistics**]: Probabilistic algorithms

General Terms

Algorithms, Theory

Keywords

Gossip, Epidemic, Asynchrony, Complexity, Adaptive versus Oblivious Adversary, Randomization, Consensus

1. INTRODUCTION

In the *gossip* problem, each process starts with an initial value, called a *rumor*, and attempts to learn all the other rumors. Gossip protocols have long been studied in various distributed computing contexts; see, for example, [11] (database consistency), [25] (failure detection) [4, 14, 17, 22] (group multicast), [21] (group membership), [8, 9] (consensus) and [20] (resource location). Almost all the theoretical research, however, has focused on synchronous systems.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.
Copyright 2008 ACM 978-1-59593-989-0/08/08 ...$5.00.

In this paper we study the efficiency of gossip in *asynchronous* systems, that is, systems in which there are no *a priori* bounds on message delay and processor speed. Processes can fail by crashing at any time, permanently halting their execution. We show how, in this context, an *adaptive adversary* can effectively limit the spread of a rumor, whereas against an *oblivious* adversary gossip can be efficiently achieved.

A simple scheme for gossiping consists of each process periodically sending its rumor—along with any new rumors that it has learned—to another randomly selected process. Such a scheme, sometimes called the *epidemic-style* paradigm for its similarity with the way diseases are spread, is very robust because of the random pattern of communication. Two natural questions, however, arise with respect to such a protocol: how often should a process transmit its rumor, and when should a process stop? In a synchronous system, both questions are readily answered: each process sends one message per round of communication, and the processes can halt (with high probability) after a sufficient number of rounds. In a seminal paper, Karp et al. [19] show that a single rumor can be disseminated in $O(\log n)$ rounds using $O(n \log\log n)$ messages, with high probability.

With further work, it is in fact possible to achieve a derandomized *deterministic* synchronous protocol, inspired by the epidemic paradigm (based on expander graphs that approximate random interactions), that needs only $O(\text{ polylog}(n))$ rounds of communication and only $O(n \text{ polylog}(n))$ messages [9], even when up to $n-1$ processes may crash. (See also [8, 16].) Perhaps unsurprisingly, globally synchronized gossip periods are key to obtaining such good performance.

While it is common to argue that distributed applications are synchronous most of the time, it is also good practice to devise algorithms that can tolerate asynchronous situations where there is no *a priori* bound on the communication delay d and the relative process speed δ. In some cases, these bounds may be unknown; in other cases, the only known bound may be very conservative, resulting in inefficient protocols; in yet other cases, there may be pathological situations in which such bounds are violated. (Think of the e-mail that took two days to reach its destination). Clearly, it is appealing to devise asynchronous gossip algorithms that do not make use of any known bound on synchrony.

The simple protocol sketched above can be engineered to work in an asynchronous environment via a simple transformation: the gossip period can be based on a local counter, rather than on bounds d and δ; every fixed number of local steps, each process sends gossip to a randomly selected process. The difficulty remains, however, to determine when to stop gossiping in order to avoid too much communication; the problem arises in part since failed processes can be confused with slow ones. Unlike in the case of a synchronous system, it is not sufficient to simply repeat the gossip step a pre-determined number of times. For example, consider the time at which two processes begin their rth iteration of gossip; because of asynchrony, for large r, it may be that one of the processes begins its rth iteration long after the other has completed that iteration. Thus if we rely on a fixed number of iterations of gossip, data may not be propagated.

The motivation of this paper is to ask whether we can devise an asynchronous gossip algorithm that tolerates $0 < f < n$ crash failures, yet performs efficiently when some bounds on d and δ indeed hold. More specifically, we are looking for asynchronous gossip algorithms with low partially synchronous complexity [12].[1]

Contributions

1. Lower Bound and Cost of Asynchrony.

Our first result demonstrates the inherent cost of asynchrony and crashes. Indirectly, this result indicates that the techniques from the synchronous world developed in [8, 9] (for example), cannot be efficiently brought to an asynchronous environment. Specifically, we show in Theorem 1 (Section 2) that any asynchronous gossip protocol—either deterministic, or against an adaptive adversary—that tolerates f faults has either $\Omega(n + f^2)$ message complexity or $\Omega(f(d+\delta))$ time complexity. Notice that the trivial gossip algorithm in which each process sends its rumor directly to everyone else has $\Theta(n^2)$ message complexity and time complexity $O(d+\delta)$. Thus, any protocol that improves on the trivial solution requires time complexity linear in f, the number of possible faults. This is in contrast to deterministic algorithms for synchronous networks that complete in only $O(\text{ polylog}(n))$ rounds using only $O(n \text{ polylog}(n))$ messages, despite tolerating $f = n-1$ failures [9].

In many ways, the lower bound is quite surprising, as epidemic-style algorithms appear relatively timing independent. Underlying our lower bound proof lies a strategy for the adversary to fight the spread of a rumor by adaptively choosing how to delay computation and when to fail processes. The strategy forces the processes to keep spreading the rumor for a long period of time, or to inflate the number of times the rumor needs to be spread.

In fact, by manipulating the relative process speeds, the adversary can trick a large number of processes into believing that the remaining processes have failed. These remaining processes are now in a quandary: if they send too many messages, then the message complexity is high; if they send too few messages, then the adversary can isolate a set of processes, resulting in a slow completion time.

We then contrast asynchronous gossip algorithms with synchronous gossip algorithms. As a corollary of our lower bound (Corollary 2), we derive the *inherent cost of asynchrony* in gossiping. Specifically, we contrast *synchronous* algorithms that know *a priori* that $d = \delta = 1$ to algorithms that are asynchronous, i.e., in which d and δ are unknown to the algorithm. We show that in the worst case, if there are f possible failures, then the most efficient asynchronous algorithm is either a factor of f slower or uses a factor of $1 + f^2/n$ more messages than the most efficient synchronous algorithm. When $f = \Theta(n)$, this implies a factor of $\Theta(n)$ loss either in time or message complexity.

2. Gossip Algorithms.

We proceed to ask whether efficient asynchronous gossip is possible in the context of an *oblivious* adversary. We present

[1] This captures the complexity of the algorithms in the subset of executions where synchrony bounds hold but are not known to the algorithm [12]. However, the algorithm is indeed asynchronous and the processes have no global clocks, nor do they manipulate the synchrony bounds. Note that it is not clear how one could measure the complexity of genuinely asynchronous executions with infinitely increasing process relative speed and communication delays.

Algorithm	Time	Messages	Model	Adversary
CK [9]	$O\left(\text{polylog}(n)\right)$	$O\left(n \text{ polylog}(n)\right)$	Synch.	Adaptive
Trivial	$O\left(d+\delta\right)$	$\Theta\left(n^2\right)$	Part. Synch.	Adaptive
Lower Bound (Section 2)	$\Omega\left(f(d+\delta)\right)$ or	$\Omega\left(n+f^2\right)$	Part. Synch.	Adaptive
EARS (Section 3)	$O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$	$O\left(n\log^3 n(d+\delta)\right)$	Part. Synch.	Oblivious
SEARS (Section 4)	$O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$	$O\left(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)}\log n(d+\delta)\right)$	Part. Synch.	Oblivious
TEARS (Section 5)	$O\left(d+\delta\right)$	$O\left(n^{7/4}\log^2 n\right)$	Part. Synch.	Oblivious

Table 1: Comparing gossip protocols for the synchronous and partially synchronous models, in the context of an adaptive or oblivious adversary.

three different algorithms that encompass different tradeoffs between time and message complexity. The results are summarized in Table 1.

The first algorithm (see Section 3), called EARS (Epidemic Asynchronous Rumor Spreading), combines a traditional epidemic-style dissemination scheme with a *progress control* scheme for collecting additional information; this additional data is necessary to decide when to stop, hence avoiding unnecessary messages. We show that this algorithm achieves $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ time complexity, and $O(n\log^3 n(d+\delta))$ message complexity, with high probability. Thus, when f is a constant fraction of n, this epidemic-style protocol is competitive with the best synchronous gossip protocols. (Note that the results in [19] refer to disseminating only a single rumor.)

Conducting the performance analysis of such an asynchronous algorithm is not straightforward; it requires examining the information *gathering* (typically found in synchronous gossip protocols), procedures like *shooting* (transmitting information from a core to the entire set of processes), and information *exchange* among pairs of processes. The technical difficulty in the analysis is related to evaluating the cost of these procedures, with respect to the unknown parameters d and δ.

The second algorithm (see Section 4), called SEARS (Spamming Epidemic Asynchronous Rumor Spreading), diverges from the pure "epidemic" style by sending more messages during each gossip period. The resulting algorithm is an asynchronous constant-time gossip algorithm with subquadratic message complexity. More specifically, we show that for every constant $\varepsilon < 1$, and for $f < n/2$, algorithm SEARS has time-complexity $O(\frac{1}{\varepsilon}(d+\delta))$ and message-complexity $O(\frac{1}{\varepsilon}n^{1+\varepsilon}\log n(d+\delta))$.

The third algorithm (see Section 5), called TEARS (Two-hop Epidemic Asynchronous Rumor Spreading), solves a weaker variant of gossip, which we call *majority gossip*, in which each process receives a majority of the rumors (rather than the rumor of each correct process). The resulting protocol achieves constant time (w.r.t. n), and a message-complexity with no dependence on d or δ, but is *stricly* subquadratic. The protocol achieves time complexity $O(d+\delta)$ and message complexity $O(n^{7/4}\log^2 n)$, for $f < n/2$.

3. Consensus.

As an application of these message-efficient gossip protocols, we present three randomized asynchronous consensus protocols. Our consensus algorithms derive from combining each of our gossip protocols with the Canetti-Rabin framework (see [6], or [2, Section 14.3]). (For consensus $f < n/2$ is assumed). The resulting protocols have time and message-complexity asymptotically equal to our gossip protocols (see Section 6); the results are summarized in Table 2. (CR-G stands for the Canetti-Rabin algorithm when used with gossip algorithm G.)

We particularly highlight the third consensus protocol as it is the first asynchronous randomized consensus algorithm that terminates in constant time (w.r.t. n) and has strictly subquadradic message-complexity. This application also motivates the further study of majority gossip, a weakening of the classic gossip problem.

To contrast our consensus algorithms to existing randomized protocols, we note that the first randomized protocol for consensus in asynchronous message-passing systems was given by Ben-Or [3]; it tolerates Byzantine failures and has exponential expected time complexity. Many other randomized algorithms have followed, considering consensus under different adversarial assumptions and failure models. See the excellent surveys of Chor and Dwork [10], Aspnes [1] and the book by Attiya and Welch [2]. To the best of our knowledge, none of the previous randomized consensus algorithms designed for an asynchronous, message-passing network achieves asymptotically subquadratic message-complexity.

More discussion and omitted proofs can be found in the full version of the paper [15].

Other Related Work.

In the context of *asynchronous networks*, Verma and Ooi [26] consider an environment that *resembles* a partially synchronous system, but assumes an *a priori* probability distribution on the communication delay; moreover, there are no crash failures. The work of Boyd et al. [5] considers gossip protocols (in the context of aggregation) in an "asynchronous" environment where local clocks are modeled as Poisson processes; there are also no crash failures in this case. Our work fundamentally differs from [26] and [5] in that we consider a fully asynchronous environment with crashes. More details on prior work on gossip in fault-prone distributed networks can be found in [24] and [18].

Algorithm	Time	Messages
Canetti-Rabin [6]	$O\left(d+\delta\right)$	$O\left(n^2\right)$
CR-EARS (Sections 3,6)	$O\left(\log^2 n(d+\delta)\right)$	$O\left(n\log^3 n(d+\delta)\right)$
CR-SEARS (Sections 4,6)	$O\left(\frac{1}{\varepsilon}(d+\delta)\right)$	$O\left(\frac{1}{\varepsilon}n^{1+\varepsilon}\log n(d+\delta)\right)$
CR-TEARS (Sections 5,6)	$O\left(d+\delta\right)$	$O\left(n^{7/4}\log^2 n\right)$

Table 2: Consensus protocols under an oblivious adversary. For consensus $f < n/2$ is assumed.

System Model

Processes.

We consider a system consisting of n message-passing, asynchronous, crash-prone processes, each with a unique identifier in a fixed set $[n] = \{1, 2, \ldots, n\}$. Up to $f < n$ processes may crash. Each process can communicate directly with all other processes; messages are not corrupted or lost in transit. The model introduced here is derived from the classical one in [12].

Timing.

For the purpose of analysis, we assume that time proceeds in discrete steps. At every time step, some arbitrary subset of the processes are scheduled to take a *local step*. In each local step: (1) a process receives some subset of the messages sent to it; (2) it performs some computation; and (3) it sends one (or more) message(s) to other process(es).

For a given execution, we define d to be the maximum delivery time of any message, and δ to be the maximum step size: if a non-failed process $\mathbf{p}$ sends a message m to process $\mathbf{q}$, and if process $\mathbf{q}$ is scheduled for a local step at any time $t' \geq t + d$, then process $\mathbf{q}$ receives message m no later than time t'; during any sequence of δ time steps, each non-crashed process is scheduled at least once. Note that in the asynchronous environment we consider, there might be no such bound d or δ in certain executions. An adversary determines the set of processes scheduled for each time step, and the set of $\leq f$ processes that crash during each time step. An *oblivious* adversary determines the schedule and failures in advance, while an *adaptive* adversary schedules and fails processes dynamically in response to the algorithm's behavior (which may depend on random choices made by the processes during the execution).

Gossip.

In this gossip problem, every process $\mathbf{p}$ begins with a rumor r_p unknown to the other processes, and maintains a collection of rumors that it has received. A gossip protocol should satisfy the following three requirements: (1) *Rumor gathering*: eventually, every correct process has added to its collection every rumor that initiated at a correct process; (2) *Validity*: if a rumor is added to a process's collection, then it is the initial rumor for some process; and (3) *Quiescence:* eventually, every process stops sending messages forever.

We say that gossip *completes* when each process has either crashed or both (a) received the rumor of every correct process and also (b) stopped sending messages. Note that it is impossible in an asynchronous system for a process to *terminate*, since a process can never be certain that it has received every correct rumor. It can, however, stop sending messages after some point.

Complexity Measures.

For a given asynchronous algorithm A, we say that A has *time complexity* $T_A^{\mathsf{asynch}}(d, \delta)$ and *message complexity* $M_A^{\mathsf{asynch}}(d, \delta)$ if for every $f < n$, for every infinite execution with bounds d and δ, every correct process completes by (expected) time $T_A^{\mathsf{asynch}}(d, \delta)$, and the (expected) number of point-to-point messages sent by all the processes combined is no more than $M_A^{\mathsf{asynch}}(d, \delta)$. For some synchronous algorithm $\widehat{A}$ in which, for every execution, $d = \delta = 1$ and this is known *a priori* by the algorithm, $T_{\widehat{A}}^{\mathsf{synch}}(d, \delta)$ and $M_{\widehat{A}}^{\mathsf{synch}}(d, \delta)$ are defined analogously. (Note that we count only the *number* of messages sent, not the total number of bits transmitted, which depends on the message size; this remains a subject for future work.)

2. THE COST OF ASYNCHRONY

We now show that no randomized gossip protocol can be both time and message efficient against an adaptive adversary. This result also establishes the cost of asynchrony: when there are $f = \Theta(n)$ possible failures, any asynchronous gossip algorithm, when compared to an optimal synchronous algorithm, either suffers a slow-down of a factor of $\Theta(n)$, or an inflation of message-complexity by a factor of $\Theta(n)$.

Underlying the lower bound lies a strategy for the adversary to fight the spreading of a rumor by adaptively choosing how to delay computations and when to fail processes. The main idea is to notice that there are two types of rumor spreading techniques: either processes send many messages in an attempt to rapidly distribute their rumors, or they rely on the cascading of messages in an attempt to send only a few. In the former case, it is easy for the adversary to construct an execution in which the protocol is not message-efficient. In the latter case, the adversary selects two processes that do not communicate directly, and prevent them from communicating by selectively failing processes that may attempt to help them. As a result, these two processes cannot terminate and hence the algorithm is slow. In both cases, we use the eventual quiescence of some of the processes to reduce the number of processes that fail in the constructed execution.

THEOREM 1. *For every gossip algorithm A, there exists $d, \delta \geq 1$ and an adaptive adversary that causes up to $f < n$ failures such that, in expectation, either: (1) $M_A^{\mathsf{asynch}}(d, \delta) = \Omega(n + f^2)$; or (2) $T_A^{\mathsf{asynch}}(d, \delta) = \Omega(f(d + \delta))$.*

PROOF. Consider some algorithm A. The $\Omega(n)$ lower bound for the number of messages is straightforward. Fix

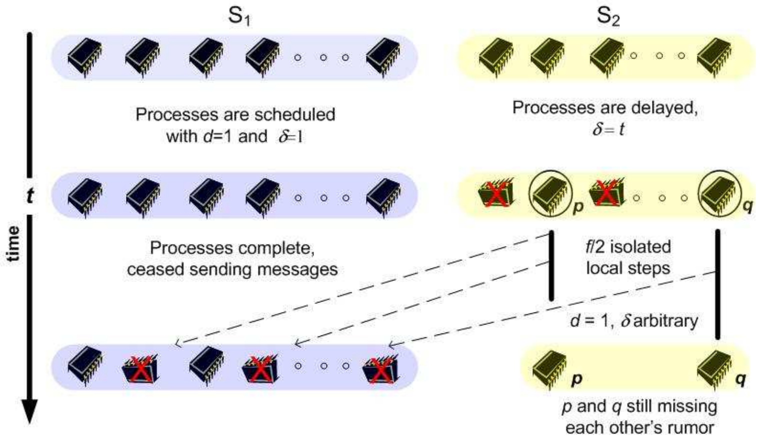

Figure 1: Illustration of the construction in Theorem 1.

$f \leq n/4$. (For $f > n/4$ the adversary follows the strategy described below with $f = n/4$.) Partition the n processes into two sets: S_1, of size $n - f/2$ and S_2, of size $f/2$. The adversary proceeds as follows: First, it executes set S_1 with $d = 1$ and $\delta = 1$ until every process in S_1 completes and ceases to send messages.

Let t be the (global) time at which this completes. Assume for the remainder of the proof that $t \leq f$; otherwise, we can fail the processes in S_2, resulting in an execution in which $d = \delta = 1$ and $t = \Omega(f(d + \delta))$. Throughout this first part, choose $\delta = f$ and schedule only processes in S_1.

Next, consider set S_2. For each process $\mathbf{p} \in S_2$, simulate the result of process $\mathbf{p}$ receiving any messages from S_1, and executing $f/2$ local steps in isolation, i.e., during which $\mathbf{p}$ receives no other messages from any other process. Since the behavior of $\mathbf{p}$ is probabilistic, this induces a distribution over the set of messages sent by $\mathbf{p}$. We say that $\mathbf{p}$ is *promiscuous* if, in expectation, $\mathbf{p}$ sends at least $f/32$ messages during the $f/2$ (isolated) local steps. Let $P \subseteq S_2$ denote the set of promiscuous processes.

There are now two cases to consider depending on number of promiscuous processes in S_2. If there are at least $f/4$ promiscuous processes ($|P| \geq f/4$), we construct an execution in which $M(d, \delta) = \Omega(f^2)$. Otherwise ($|P| < f/4$), we construct an execution in which $T(d, \delta) = \Omega(f(d + \delta))$.

Case 1: $|P| \geq f/4$. Then, after time t, the adversary schedules all of the processes in S_2 in each of the following $f/2$ time steps ($\delta = 1$). The adversary ensures that none of these messages are delivered, i.e., $d \geq f/2 + 1$. Thus, the $f/4$ promiscuous processes send, in expectation, $f/32$ messages each (and receive no messages), resulting in an expected message complexity $M_A^{\mathsf{asynch}}(d, \delta) = \Omega(f^2)$, as desired. Notice that in this case, the adversary does not fail any processes.

Case 2: $|P| < f/4$. Let $S = S_2 \setminus P$ (the set of non-promiscuous processes), and let $\nu = |S|$. The adversary proceeds to identify two non-promiscuous processes that have a constant probability of not communicating with each other; all other processes in S_2 are failed. (See Figure 1 for an illustration.)

In order to identify two such non-promiscuous processes, for each non-promiscuous $\mathbf{p} \in S$, we define the set $N(\mathbf{p})$ to be the set of processes that $\mathbf{p}$ sends one or more messages to with probability $< 1/4$ during $f/2$ (isolated) local steps. (If $\mathbf{p}$ sends more than one message to the same process, then $N(\mathbf{p})$ requires that the the probability to send one message plus the probability to send two messages and so on is less than $1/4$). We know, since $\mathbf{p}$ is non-promiscuous, that the expected number of messages sent by $\mathbf{p}$ is $< f/32$. Thus, $|N(\mathbf{p})| > 7 \cdot f/8$: if not, then there exist (at least) $f/8$ processes that $\mathbf{p}$ sends (at least) one message with probability $\geq 1/4$, implying that in expectation $\mathbf{p}$ sends at least $f/32$ messages, resulting in a contradiction.

Next, since at most $f/8$ processes are not in $N(\mathbf{p})$, and since there are at least $f/4$ non-promiscuous processes ($\nu = |S_2| - |P|$, $|S_2| = f/2$, $|P| < f/4$), we conclude that there are at least $\nu/2$ non-promiscuous processes in $N(\mathbf{p})$. Thus, for each non-promiscuous $\mathbf{p} \in S_2$, there are at least $\nu/2$ non-promiscuous processes in S_2 that are sent a message by $\mathbf{p}$ with probability $< 1/4$.

We claim, then, that there exist two non-promiscuous processes $\mathbf{p}, \mathbf{q} \in S_2$ such that $\mathbf{q} \in N(\mathbf{p})$ and $\mathbf{p} \in N(\mathbf{q})$: Consider the (logical) directed graph on ν non-promiscuous nodes in which there is an edge from $\mathbf{p}$ to $\mathbf{q}$ if $\mathbf{q} \in N(\mathbf{p})$. Since each $\mathbf{p}$ has $\nu/2$ outgoing edges, there are a total of $\nu^2/2$ edges in the graph. However, there are only $\binom{\nu}{2} = \nu(\nu-1)/2$ pairs of nodes in the graph, implying that there must exist a pair of nodes with edges in both directions, as required.

The adversary fails all the nodes in S_2 except $\mathbf{p}$ and $\mathbf{q}$ immediately at time t, prior to taking any local steps. The adversary then executes $\mathbf{p}$ and $\mathbf{q}$ for $f/2$ local steps, delivering all messages with delay 1, i.e., $d = 1$. Since $\mathbf{p}$ and $\mathbf{q}$ have not coordinated via previous messages, they choose to send their messages independently, and thus we have established that with probability at least $(1 - 1/4)(1 - 1/4) = 9/16$, $\mathbf{p}$ does not send a message to $\mathbf{q}$ and $\mathbf{q}$ does not send a message to $\mathbf{p}$. The adversary fails every other process in S_1 to which $\mathbf{p}$ and $\mathbf{q}$ send a message (notice that processes in S_2 have already been failed). Since $\mathbf{p}$ and $\mathbf{q}$ are not promiscuous, each in expectation sends no more than $f/32$ messages. By Markov's inequality, we conclude that each, with probability at least $3/4$, sends no more than $f/8$ messages. (Let X be the random variable for the number of messages sent by $\mathbf{p}$. Then, $\Pr(X \geq f/8) \leq \frac{f/32}{f/8} = 1/4$. Hence, $\Pr(X < f/8) > 3/4$.) Thus, since the processes are independent, with probability $9/16$, the two processes $\mathbf{p}$ and $\mathbf{q}$ together send at most $f/4$

messages, resulting in the total number of failed processes being no more than $3f/4 - 2 < 3f/4 < f$: $f/4$ processes in S_1 and $f/2 - 2$ processes in S_2.

Finally, using a union bound, we observe that $\mathbf{p}$ and $\mathbf{q}$ do not communicate with each other, and do not send more than $f/4$ combined messages, with probability at least $(1 - (7/16 + 7/16)) = 1/8$. We also notice that in this case, $\mathbf{p}$ and $\mathbf{q}$ cannot terminate during the $f/2$ (isolated) local steps: they have not received each other's rumors. Since in this case, $d = 1$ and each local step takes time δ, we conclude that $\mathbf{p}$ and $\mathbf{q}$ run for time at least $(d+\delta)f/2$ with probability at least $1/8$. Thus, in expectation, $T_A^{\mathsf{asynch}}(d, \delta) = \Omega(f(d+\delta))$, as desired. □

As a corollary, we consider the worst-case ratio of the cost of asynchronous and synchronous algorithms. For a given asynchronous algorithm A, we define the time and message cost-of-asynchrony (CoA) as follows:

$$T(A)_{CoA} = \max_{d,\delta} \left(\frac{T_A^{\mathsf{asynch}}(d,\delta)}{\min_{\widehat{A}} T_{\widehat{A}}^{\mathsf{synch}}(d,\delta)} \right)$$

$$M(A)_{CoA} = \max_{d,\delta} \left(\frac{M_A^{\mathsf{asynch}}(d,\delta)}{\min_{\widehat{A}} M_{\widehat{A}}^{\mathsf{synch}}(d,\delta)} \right).$$

We conclude from Theorem 1 that there is an inherent cost to tolerating asynchrony. In particular, the most efficient asynchronous gossip algorithms are significantly less efficient than the most efficient synchronous gosssip algorithms:

Corollary 2 (Cost of Asynchrony). *For every asynchronous gossip algorithm A, either:*

$$T(A)_{CoA} = \Omega(f)$$

or

$$M(A)_{CoA} = \Omega(1 + f^2/n).$$

3. EFFICIENT EPIDEMIC GOSSIP

In this section, we present an epidemic-style asynchronous gossip algorithm, called EARS (Epidemic Asynchronous Rumor Spreading), that can tolerate up to $f < n$ failures. We then show in Section 3.2 that in the context of an oblivious adversary, the algorithm is both time and message efficient, achieving $O(\frac{n}{n-f} \log^2 n(d+\delta))$ time complexity and $O(n \log^2 n(d+\delta))$ message complexity.

3.1 Algorithm EARS

The algorithm presented in this section is based on the well-known epidemic paradigm, augmented to maintain and propagate additional information about the ongoing progress in distributing the rumors. In each step, a process chooses a target at random and sends it all the information that it has collected. This procedure is devised to achieve three properties: (1) *gathering*: after some period of time, every rumor originating at a correct process is known to every process in a large core of correct processes; (2) *shooting*: every so often, every rumor known to the large core is sent to every other process in the system; (3) *exchange*: every so often, every pair of correct processes in the core exchange information about who has been shot.

```
EARS($r_p$)
  ▷ Initialization:
  $V(\mathbf{p}) \leftarrow \{r_p\}$          ▷ a set of processes
  $I(\mathbf{p}) \leftarrow \emptyset$            ▷ a set of pairs $\langle r, \mathbf{p} \rangle$
  $L(\mathbf{p}) \leftarrow [n]$           ▷ a set of processes
  $sleep\_cnt \leftarrow 0$         ▷ an integer

  repeat
     for every message $m = \langle V, I \rangle$ received do
           $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \cup m.V$
           $I(\mathbf{p}) \leftarrow I(\mathbf{p}) \cup m.I$
     Update $L(\mathbf{p})$ based on $V(\mathbf{p})$ and $I(\mathbf{p})$.
     if $L(\mathbf{p}) = \emptyset$
           then    $sleep\_cnt \leftarrow sleep\_cnt + 1$
           else    $sleep\_cnt \leftarrow 0$
     if $sleep\_cnt < \Theta\left(\frac{n}{n-f} \log n\right)$ then
           ▷ Epidemic Transmission Mode:
           Choose $\mathbf{q}$ uniformly at random from $[n]$.
           Send $m = \langle V(\mathbf{p}), I(\mathbf{p}) \rangle$ to process $\mathbf{q}$.
           for every $r \in V(\mathbf{p})$ do
              $I(\mathbf{p}) \leftarrow I(\mathbf{p}) \cup (r, \mathbf{q})$
           Update $L(\mathbf{p})$ based on $V(\mathbf{p})$ and $I(\mathbf{p})$.
```

Figure 2: The Epidemic-style gossip algorithm EARS, stated for process p; r_p denotes the rumor of p. Every time p is scheduled to take a step, it executes one iteration of the main loop.

In more detail, the algorithm proceeds as follows. (Pseudocode is presented in Figure 2.) Each process $\mathbf{p}$ maintains a set $V(\mathbf{p})$ containing all the rumors known to $\mathbf{p}$. Initially $V(\mathbf{p})$ contains only $\mathbf{p}$'s initial rumor. Each process also maintains an informed-list $I(\mathbf{p})$ which contains pairs of rumors and processes: when $(r, \mathbf{q}) \in I(\mathbf{p})$, this implies that $\mathbf{p}$ knows that rumor r has been sent to process $\mathbf{q}$ by some process.

In each local step, a process sends a message containing $V(\mathbf{p})$ and $I(\mathbf{p})$ to a process $\mathbf{q}$ chosen uniformly at random from $[n]$. Process $\mathbf{p}$ then adds all pairs $(r, \mathbf{q})$, for $r \in V(\mathbf{p})$, to the informed-list $I(\mathbf{p})$. When a process $\mathbf{q}$ receives a message from $\mathbf{p}$, it updates its local sets $V(\mathbf{q})$ and $I(\mathbf{q})$.

Let $L(\mathbf{p})$ be the set of processes that $\mathbf{p}$ cannot ascertain (via $I(\mathbf{p})$) whether they have been sent every rumor in $V(\mathbf{p})$, i.e., $L(\mathbf{p}) = \{\mathbf{q} : \exists r \in V(\mathbf{p}), (r, \mathbf{q}) \notin I(\mathbf{p})\}$. When $L(\mathbf{p})$ is empty for process $\mathbf{p}$, i.e., when every rumor known to $\mathbf{p}$ has been sent to every process, $\mathbf{p}$ enters the shut-down phase during which $\mathbf{p}$ ensures that its informed-list $I(\mathbf{p})$ has been disseminated sufficiently. This phase lasts for $\Theta(\frac{n}{n-f} \log n)$ local steps; during the shut-down phase, $\mathbf{p}$ continues as before, receiving messages from other processes and sending a *shut-down* message to a process chosen uniformly at random from $[n]$. After the shut-down phase completes, $\mathbf{p}$ stops sending messages and sleeps.

During the shut-down phase, and afterwards while $\mathbf{p}$ is asleep, it may continue to receive and process messages from other processes that are still awake. If $\mathbf{p}$ receives a new rumor that has not yet been sent to some process, i.e., if $L(\mathbf{p})$ becomes non-empty, then $\mathbf{p}$ aborts the shut-down phase or awakens and resumes the normal epidemic process until $L(\mathbf{p})$ becomes empty again.

3.2 Analysis of Algorithm EARS

In this section, we argue that with an oblivious (d, δ)-adversary, the algorithm completes (correctly) by time $O(\frac{n}{n-f}\log^2 n(d+\delta))$ and sends $O(n\log^3 n(d+\delta))$ messages, with high probability.

For the purposes of the analysis we partition the execution into epochs such that in each epoch, only a constant fraction of the processes fail. Since at most f processes crash in an execution, there are at most $\log\frac{n}{n-f}$ such epochs. We identify the first epoch i of length $\Theta(2^i\log^2 n(d+\delta))$; a simple counting argument suffices to show that this occurs no later than time $\Theta(\frac{n}{n-f}\log^2 n(d+\delta))$. Let A be the set of processes that are non-faulty in epoch i; by assumption, we have $n/2^{i+1} < |A| \le n/2^i$. Each epoch is partitioned into 7 *stages*, each of length $\Theta(2^i\log^2 n(d+\delta))$. Each correct process thus takes at least $\Theta(2^i\log n)$ local steps in each stage.

At a high level, the proof proceeds to show the following sequence of claims, each of which corresponds to a stage: (1) Processes in A collectively gather all the rumors in the system. (2) Processes in A exchange information, ensuring that every process in A knows all the rumors. (3) Processes in A collectively send all the rumors to every other process. (4) Processes in A exchange information, ensuring that every process in A knows that all the rumors have been sent. (5) Processes in A enter the shut-down phase. (6) Processes in A collectively notify all the processes in the system that it is time to shut down. (7) Every process sleeps.

The first key lemma shows that in each stage of epoch i, all the processes in i exchange information. This resembles the analysis of typical epidemic-style algorithms, with the additional complication that some processes may be sleeping (believing perhaps incorrectly that the gossip is complete).

LEMMA 3 (EXCHANGE PROPERTY). *For every process* $\mathbf{p} \in A$ *and stage* j *of epoch* i*:*
(1) All rumors known by $\mathbf{p}$ *at the beginning of stage* j *are known to every* $\mathbf{q} \in A$ *at the end of* j*, w.h.p.*
(2) If no process in A *is asleep by the end of stage* j*, then all* pairs *known to* $\mathbf{p}$ *in* $I(\mathbf{p})$ *at the beginning of stage* j *are known to all other processes in* A *at the end of stage* j*, w.h.p.*

PROOF (SKETCH). We begin with Part (2). Define sets B_k, for $0 \le k \le \log|A|$: Let B_0 contain processes in A that know the rumors and pairs from process $\mathbf{p}$ at the beginning of stage j; let B_{k+1} contain processes in $A \setminus (B_0 \cup \ldots \cup B_k)$ that were sent a message from some process $\mathbf{q}$ in B_k in the first $c \cdot 2^i\log n - 1$ local steps after $\mathbf{q}$ has received information about $\mathbf{p}$'s rumors and pairs at the beginning of phase j.

As long as $|B_k| < |A|/8$, the sets B_k grow at least exponentially in k with high probability. In the case where $|B_k| > |A|/8$, there are a sufficient number of processes in B_k to ensure that every other process in A is sent a message containing information on $\mathbf{p}$'s rumors and pairs at the beginning of stage j, with high probability. The same argument holds with respect to Part (1), with the additional complication that if a process involved in the dissemination is asleep, then its data has already been disseminated. □

We now argue that every process in A learns every possible rumor by the end of stage 1. This lemma follows by counting the number of messages sent by any correct process in stage 1, ensuring that each rumor is received by some process in A; we then apply Lemma 3. Let $V_{\mathtt{all}}$ be the set of rumors that are eventually learned by some correct process. (Since every correct process "learns" its own rumor, it is sufficient to ensure that every correct process learns $V_{\mathtt{all}}$.)

LEMMA 4 (GATHERING PROPERTY). *At the end of stage* 2*, for every non-failed* $\mathbf{p} \in A$*,* $V_{all} \subseteq V(\mathbf{p})$ *w.h.p.*

The key remaining claim is that every process sleeps by the end of epoch i and never awakes thereafter. We begin by observing: since every process in A has already learned every rumor in $V_{\mathtt{all}}$by the end of stage 2 (w.h.p.), we can be certain that no process in A learns any new rumors at any later point.

Next, we note that by the end of stage 3, every rumor in $V_{\mathtt{all}}$ has been sent to every process in $[n]$. This follows by counting the number of messages sent by (non-sleeping) processes in A:

LEMMA 5 (SHOOTING PROPERTY). *For every process* $\mathbf{q} \in [n]$*, there exists some process* $\mathbf{p} \in A$ *such that at the end of stage 3 in epoch* i*,* $\mathbf{q} \notin L(\mathbf{p})$ *with high probability.*

It therefore easy to see that by the end of stage 4, as a result of Lemma 5 and Lemma 3, at least one process in A has entered the shut-down phase. Since information is rapidly exchanged, it then follows that soon thereafter every process in A enters the shut-down phase, followed by every correct process in $[n]$; it requires some care, however, to ensure that processes do not go to sleep prior to propagating the shut-down information. (The argument is somewhat analogous to Lemma 3, in that we maintain increasing sized sets of nodes that have entered the shut-down phase, but not yet gone to sleep.) We now conclude:

THEOREM 6. *Algorithm* EARS *completes gossip with time complexity* $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ *and message complexity* $O(n\log^3 n(d+\delta))$*, with high probability under an oblivious adversary.*

PROOF (SKETCH).. By Lemma 4, we know that every rumor in $V_{\mathtt{all}}$ has been learned by every process in A, and thus the gossip protocol succeeds (w.h.p.). By Lemmas 5 and 3, it follows that some process in A has entered the shut-down phase by the end of stage 4 with high probability. From this and the rapid information exchange (Lemma 3), it follows that all processes in A have entered the shut-down phase by the end of stage 6, and we have already observed that, no process exits the shut-down phase at this point. Hence by the end of stage 7 every process has gone to sleep with high probability. Epoch i ends no later than time $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$, resulting in the desired time complexity with high probability.

We now calculate the number of messages sent. In each epoch $k < i$ there are at most $n/2^k$ non-failed processes. Since epoch i is the first "long" epoch, we know that each process that is alive at the beginning of epoch k sends at most $O(2^k\log^2 n(d+\delta))$ messages in epoch k. Thus, in epoch k, non-failed processes send at most $O(n\log^2 n(d+\delta))$ messages. The accounting for epoch i is similar, since every process sleeps at the end of stage 7. Since there are at most $\log n$ epochs, the result follows. □

4. CONSTANT-TIME GOSSIP

Algorithm EARS can be modified to yield a constant-time randomized asynchronous gossip algorithm, which we call

SEARS (Spamming Epidemic Asynchronous Rumor Spreading). The only difference in the algorithm is as follows:

- In each local step, each process sends messages to $\Theta(n^{\varepsilon} \log n)$ processes chosen at random.
- Each process takes only one shut-down step.

Then, we get the following.

THEOREM 7. *For every constant $\varepsilon < 1$, algorithm* SEARS *has time-complexity $O(\frac{n}{\varepsilon(n-f)}(d+\delta))$ and message-complexity $O(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)} \log n(d+\delta))$.*

Notice that for $f < n/2$ or f/n constant, the above results in a constant-time (w.r.t. n) gossip protocol with subquadratic message-complexity.
The correctness follows from the following sequence of facts: Consider some particular rumor r that begins at some correct process $\mathbf{p}$. Within $O(\frac{n}{n-f}(d+\delta))$ time, process $\mathbf{p}$ has sent its rumor to at least $\Theta(n^{\varepsilon})$ correct processes (with high probability). In every further $O(\frac{n}{n-f}(d+\delta))$ time, the number of correct processes that know r grows by a factor of n^{ε}; after $1/\varepsilon$ steps, a constant faction of the correct nodes know r (with high probability). Within a further $O(\frac{n}{n-f}(d+\delta))$ time, these correct processes have, collectively, sent r to every process in the system (with high probability). Then, in the same way that rumor r spreads, the fact that "r has been sent to every process" spreads, reaching every correct process within $O(\frac{n/\varepsilon}{n-f}(d+\delta))$ time, after which there is one iteration of shut-down messages followed by termination (with high probability).

5. CONSTANT-TIME MAJORITY GOSSIP

The previous gossip protocols ensure that eventually every correct rumor is disseminated. However, for the purpose of various applications, including consensus [6] and do-all [7], it suffices to require that each correct process receives only a majority of the rumors (rather than receiving the rumor of each correct process). We refer to this weaker version of gossip as *majority gossip*. By restricting our attention to the problem of majority gossip, we devise a gossip protocol, called TEARS (Two-hop Epidemic Asynchronous Rumor Spreading), that completes in $O(d+\delta)$ time with message complexity $O(n^{7/4} \log^2 n)$, with high probability. Notice that the message complexity does not depend on d and δ, i.e., it is *strictly* sub-quadratic. In order to make majority gossip feasible, we need to assume that $f < n/2$.

5.1 Algorithm TEARS

We describe algorithm TEARS from the point of view of a process $\mathbf{p}$. (Pseudocode is presented in Figure 3.) We define three additional parameters to simplify the description of the algorithm: $a = 4\sqrt{n} \log n$, $\mu = \frac{a}{2}$, and $\kappa = 8n^{1/4} \log n$.[2] Additionally, each process $\mathbf{p}$ selects locally two subsets of processes $\Pi_1(\mathbf{p}), \Pi_2(\mathbf{p})$ in such a way that every other process $\mathbf{q}$, where $\mathbf{q} \neq \mathbf{p}$, is included in set $\Pi_1(\mathbf{p})$ (or in set $\Pi_2(\mathbf{p})$, resp.) with probability a/n, independently at random. In the first local step, each process $\mathbf{p}$ sends a message, containing its own rumor and a flag raised up, to all processes in $\Pi_1(\mathbf{p})$. We call such messages *first-level messages*.

```
TEARS($r_p$)
▷ Initialization:
$a \leftarrow 4\sqrt{n}\log n$
$\mu \leftarrow 2\sqrt{n}\log n$
$\kappa \leftarrow 8\sqrt[4]{n}\log n$
$V(\mathbf{p}) \leftarrow \{r_p\}$
$\Pi_1(\mathbf{p})$ : $\forall \mathbf{q} \neq \mathbf{p}$ put $\mathbf{q}$ into $\Pi_1(\mathbf{p})$ with probability $a/n$
$\Pi_2(\mathbf{p})$ : $\forall \mathbf{q} \neq \mathbf{p}$ put $\mathbf{q}$ into $\Pi_2(\mathbf{p})$ with probability $a/n$
$flag \leftarrow \uparrow$
$up\_msg\_cnt \leftarrow 0$

repeat
    ▷ Two-Hop Transmission
    for every process $\mathbf{q} \in \Pi_1(\mathbf{p})$ do
            send $m = \langle V(\mathbf{p}), flag \rangle$ to $\mathbf{q}$
    $flag \leftarrow \downarrow$
    for every message $m$ received do
            $V(\mathbf{p}) = V(\mathbf{p}) \cup m.V$
            if $m.flag = \uparrow$
               then $up\_msg\_cnt \leftarrow up\_msg\_cnt + 1$
            $bcast \leftarrow$ false
            if $(\mu - \kappa \leq up\_msg\_cnt < \mu + \kappa)$
               then $bcast \leftarrow$ true
            if $(up\_msg\_cnt - \mu)/\kappa$ is a positive integer
               then $bcast \leftarrow$ true
            if $bcast =$ true
               then for every process $\mathbf{q} \in \Pi_2(\mathbf{p})$ do
                        send $m = \langle V(\mathbf{p}), flag \rangle$ to $\mathbf{q}$
```

Figure 3: Two-hop majority gossip algorithm TEARS, stated for process p; r_p denotes the rumor of p.

After receiving $\mu - \kappa$ first-level messages, each process $\mathbf{p}$ sends a *second-level message*, that is, a message consisting of all gathered rumors, to all processes in set $\Pi_2(\mathbf{p})$. It does the same after receiving $\mu + j$ first-level messages, for every $-\kappa < j < \kappa$, and later after receiving $\mu + i\kappa$ first-level messages, for every positive integer i.

Notice that unlike algorithm EARS, a process does not send a message in every step; instead, a process sends messages based on how many first-level message have been received.

5.2 Analysis of Algorithm TEARS

First observe the following estimates for the number of messages sent by processes in a single step.

LEMMA 8. *Every process sends either 0 or between $a - \kappa$ and $a + \kappa$ point-to-point messages in each step, with probability at least $1 - 2/n^3$.*

For each process $\mathbf{p}$, let $S_{\mathbf{p}}$ consist of the local steps of process $\mathbf{p}$ before sending its last second-level message; we call it the *safe epoch of process* $\mathbf{p}$. Note that process $\mathbf{p}$ may receive some first-level messages after that time, but it does not re-send these in any second-level message. We say that the rumor of process $\mathbf{q}$ is *safe in process* $\mathbf{p}$ if $\mathbf{p}$ receives the rumor of $\mathbf{q}$ in some first-level message that arrives in its safe period. If a rumor is safe in at least $\sqrt{n}$ non-faulty processes then we call such rumor *well-distributed*. The intuition behind a safe rumor is that it will be sent by the host to a random processes as a part of some second-level message,

[2] We also assume that n is sufficiently large; otherwise the asymptotic complexities are all constants.

unless the host becomes faulty, and therefore a rumor that is safe in sufficiently many processes will eventually reach every non-faulty process.

We perform the analysis of correctness in three steps, captured by the three following lemmas, each proved to hold with high probability. The first result explores the obliviousness of the adversary. It shows that *almost* a majority of processes cannot be sufficiently disturbed by an oblivious adversary during the first hop of random rumor spreading.

Lemma 9. *There are at least $n/2 - \frac{n}{\log n}$ well-distributed rumors, with probability at least $1 - 2/n^2$.*

The next lemma formalizes the probabilistic intuition that any rumor well-distributed after the first hop is successfully delivered to all non-faulty processes in the second hop.

Lemma 10. *Each well-distributed rumor is eventually received by each non-faulty process, with probability at least $1 - 3/n^2$.*

It can be shown that there are a large number of rumors that are not well-distributed after the first hop. Moreover, each non-faulty process receives in its second-level messages a number of such rumors that complements the number of well-distributed ones, reaching a majority of all rumors.

Lemma 11. *Every non-faulty process receives such a number of non well-distributed rumors that complements the number of well-distributed rumors received to reach a majority of all rumors, with probability at least $1 - 4/n^2$.*

Theorem 12. *Algorithm* tears *completes majority gossip with time complexity $O(d + \delta)$ and message complexity $O(n^{7/4} \log^2 n)$, with high probability under an oblivious adversary.*

Proof. The correctness is guaranteed by Lemmas 9, 10 and 11 with probability at least $1 - \frac{2}{n^2} - \frac{3}{n^2} - \frac{4}{n^2} \geq 1 - \frac{9}{n^2}$. It remains to prove the complexity bounds.
Time complexity. All first-level messages arrive by time $d+\delta$, by time $2d + \delta$ every second-level message is sent, and it arrives by time $2d + 2\delta$. Hence, the bound $O(d + \delta)$ follows.
Message complexity. Each non-faulty process sends $a + \kappa \leq 10\sqrt{n} \log n$ first-level messages and thus receives at most $40\sqrt{n} \log n$ first-level point-to-point messages with high probability (by Lemma 8, Chernoff bound and the upper bound $n/2$ on the number of failures). Therefore it sends less than

$$\left(2\kappa + 1 + \frac{40\sqrt{n}\log n}{\kappa}\right) \cdot (a + \kappa) = O(n^{3/4} \log^2 n)$$

second-level point-to-point messages, all with high probability (again by Lemma 8 and a union bound applied to this and the previous events). By applying union bound to all processes, the bound $O(n^{7/4} \log^2 n)$ follows, also with high probability. □

6. RANDOMIZED CONSENSUS

In this section, we show how we implement message-efficient fault-tolerant consensus, based on the gossip protocols presented in Sections 3–5. Recall that consensus consists of n nodes, each with an initial value v_i, trying to choose an output (i.e., *decision*) satisfying: (1) *Agreement*: Every value output by a process is the same. (2) *Validity*: Every value output is some process's initial value. (3) *Termination*: Every process eventually outputs a value, with high probability. The key contribution of this section is showing how gossip (and majority gossip) can be used in the context of the Rabin-Canetti framework to produce an efficient consensus protocol.

We begin by recalling the Rabin-Canetti framework introduced by [6], and we follow the simplified presentation of [2, Section 14.3] for crash-prone networks. Throughout the protocol, each process repeats three rounds of voting until a decision is reached: (1) Each process votes on its *estimate* (originally, its initial value); if one estimate receives all the votes, then that value is decided; if some estimate is voted on by a majority, then that estimate is *preferred*. (2) In the second election, each process votes on its preferred value; if everyone votes to prefer the same value, then that value is adopted as the estimate; otherwise, a process proceeds to the third round of voting which simulates a shared random coin. (See [2] for more details.)

Voting is implemented by a routine get-core which exchanges information among the processes. It returns a set of votes to each participant satisfying the following: there exists some set S containing at least a majority of the votes such that each call to get-core returns at least the votes in S. As presented in [2], get-core is implemented by three sequential phases of all-to-all communication in which each process sends all the votes it has received in that round to everyone else, leading to $O(n^2)$ message complexity.

Efficient (majority) gossip can be used to reduce the message complexity. Specifically, we implement get-core via three sequential instances of asynchronous (majority) gossip, each of which terminates when a process receives $\lfloor n/2 \rfloor + 1$ rumors. Notice, though, that gossip is initiated here asynchronously; previously, we had assumed that gossip began simultaneously. Assume that as soon as a process receives a gossip message, it begins to gossip itself. If any process begins gossip using algorithm ears then within time $O((d + \delta) \log^2 n)$, every non-failed process begins to gossip; this follows immediately from the epidemic spread of the initiator's rumor. Similarly, with algorithm sears (resp. tears), every non-failed process begins to gossip within $O(\frac{1}{\varepsilon}(d + \delta))$ time (resp. $O(d + \delta)$). Thus, asynchronous gossip initiation has no asymptotic impact on time or message complexity.

It remains to ensure that each process begins to gossip immediately upon receiving a gossip message. In order to achieve this, each gossip message includes a history of all prior completed calls to gossip and get-core. As soon as a process receives a gossip message, it can use the received history log to "catch up" with the sender of that message, adopting the sender's outcome for each completed gossip and get-core.

From the above we conclude the following theorem:

Theorem 13. *For an oblivious adversary and a minority of failures, in expectation,*

- *Algorithm CR-*ears *has $O\left(\log^2 n(d + \delta)\right)$ time and $O(n \log^3 n(d + \delta))$ message complexity;*
- *Algorithm CR-*sears *has, $\forall \varepsilon < 1$, $O(\frac{1}{\varepsilon}(d + \delta))$ time and $O(n^{1+\varepsilon} \log n(d + \delta))$ message complexity;*
- *Algorithm CR-*tears *has $O(d + \delta)$ time and $O(n^{7/4} \log^2 n)$ message complexity;*

7. CONCLUSIONS

In this paper, we have initiated the study of the complexity of gossip in an asynchronous, message-passing distributed system subject to processes crash failures. Our results demonstrate that gossip is inherently inefficient in the context of an adaptive adversary, but that it is possible to develop efficient, randomized, asynchronous gossip algorithms under an oblivious adversary. Our gossip algorithms can be used to implement efficient asynchronous randomize consensus protocols; one variant terminates in constant time and has strictly subquadratic messsage complexity. This last result is achieved by considering a weaker version of gossip, called majority gossip.

One interesting open question is whether we can achieve even better message complexity for a constant-round consensus protocol; the key is to improve the message complexity of majority gossip. Another interesting question is whether majority gossip is any easier than general gossip; as of yet, there no known lower bounds for majority gossip. Or, does there exist an efficient deterministic asynchronous algorithm for majority gossip problem. We believe that efficient solutions to majority gossip can lead to efficient solutions for other distributed problems, even beyond consensus, such as load balancing and distributed atomic shared memory implementations. Finally, we believe it is interesting to investigate the bit complexity of asynchronous gossip (that is, the total number of bits exchanged in a given computation).

8. REFERENCES

[1] J. Aspnes, Randomized protocols for asynchronous consensus, *Distributed Computing*, 16(2–3) (2003) 165–175.

[2] H. Attiya and J. Welch, *“Distributed Computing: Fundamentals, Simulations, and Advanced Topics,”* Wiley-Interscience, Hoboken, NJ, 2004.

[3] M. Ben-Or, Another advantage of free choice: Completely asynchronous agreement protocols, in *Proc. of 2nd ACM Symposium on Principles of Distributed Computing (PODC)*, 1983, pp. 27–30.

[4] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, Bimodal multicast, *ACM Trans. on Computer Systems*, 17(2) (1999) 41–86.

[5] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, Randomized gossip algorithms, *IEEE Transactions on Information Theory*, 52(6) (2006) 2508–2530.

[6] R. Canetti and T. Rabin, Fast asynchronous Byzantine agreement with optimal resilience, in *Proc. of 25th ACM Symposium on Theory of Computing (STOC)*, 1993, pp. 42-51.

[7] B.S. Chlebus, L. Gasieniec, D.R. Kowalski, and A.A. Shvartsman, Bounding work and communication in robust cooperative computation, in *Proc. of the 16th International Symposium on Distributed Computing (DISC)*, 2002, pp. 295–310.

[8] B.S. Chlebus and D.R. Kowalski, Robust gossiping with an application to consensus, *Journal of Computer and System Sciences*, 72 (2006) 1262–1281.

[9] B.S. Chlebus and D.R. Kowalski, Time and communication efficient consensus for crash failures, in *Proc. of 20th International Symposium on Distributed Computing (DISC)*, 2006, pp. 314–328.

[10] B. Chor and C. Dwork, Randomization in Byzantine agreement, *Advances in Computing Research 5: Randomness and Computation*, (1989) pp. 443–497.

[11] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, Epidemic algorithms for replicated database maintenance, in *Proc. of the 6th ACM Symposium on Principles of Distributed Computing (PODC)*, 1987, pp. 1–12.

[12] C. Dwork, N. Lynch, and L. Stockmeyer, Consensus in the presence of partial synchrony, *Journal of ACM*, 35(2) (1988) 288–323.

[13] M. Fisher, N. Lynch, and M. Paterson, Impossibility of distributed consensus with one faulty process, *Journal of the ACM*, 32 (1985) 374–382.

[14] P. Eugster, R. Guerraoui, S. Handurukande, A-M Kermarrec, and P. Kouznetsov, Lightweight probabilistic broadcast, *ACM Transactions on Computer Systems*, 21(4) (2003).

[15] C. Georgiou, S. Gilbert, R. Guerraoui, and D.R. Kowalski, On the complexity of asynchronous gossip, Technical Report, 2008.

[16] C. Georgiou, D.R., Kowalski, and A.A. Shvartsman, Efficient gossip and robust distributed computation, *Theoretical Computer Science*, 347 (2005) 130–166.

[17] I. Gupta, A.M. Kermarrec, and A.J. Ganesh, Efficient epidemic-style protocols for reliable and scalable multicast, in *Proc. of 21st IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2002.

[18] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzika, and W. Unger, *“Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance,”*, Springer-Verlag, 2005.

[19] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, Randomized Rumor Spreading, in *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pp. 565–574.

[20] D. Kempe, J. Kleinberg, and A. Demers, Spatial gossip and resource location protocols, *Journal of ACM*, 51 (2004) 943–967.

[21] A. Kermarrec, L. Massoulie, and A. Ganesh, Probabilistic reliable multicast in ad hoc networks, *IEEE Trans. on Parallel and Distr. Syst.*, 14 (2003).

[22] J. Luo, P. Eugster, and J.P. Hubaux, Route driven gossip: Probabilistic reliable multicast in ad hoc networks, in *Proc. of 21st IEEE Conference on Computer Communications (INFOCOM)*, 2003.

[23] M. Mitzenmacher and E. Upfal, *“Probability and Computing,”* Cambridge University Press, 2005.

[24] A. Pelc, Fault-tolerant broadcasting and gossiping in communication networks, *Networks*, 28 (1996) 143–156.

[25] R. van Renesse, Y. Minsky, and M. Hayden, A gossip-style failure detection service, in *Proc. of IFIP Int-l Conference on Distributed Systems Platforms and Open Distributed Processing*, 1998, pp. 55–70.

[26] S. Verma and W.T. Ooi, Controlling gossip protocol infection pattern using adaptive fanout, in *Proc. of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2005, pp. 665–674.