Asynchronous Gossip

CHRYSSIS GEORGIOU, University of Cyprus
SETH GILBERT, National University of Singapore
RACHID GUERRAOUI, École Polytechnique Fédérale de Lausanne
DARIUSZ R. KOWALSKI, University of Liverpool

We study the complexity of *gossip* in an asynchronous, message-passing fault-prone distributed system. We show that an *adaptive* adversary can significantly hamper the spreading of a rumor, while an *oblivious* adversary cannot. The algorithmic techniques proposed in this article can be used for improving the message complexity of distributed algorithms that rely on an all-to-all message exchange paradigm and are designed for an asynchronous environment. As an example, we show how to improve the message complexity of asynchronous randomized consensus.

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 (Including Monte Carlo)*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Gossip, epidemic, asynchrony, complexity, adaptive versus oblivious adversary, randomization, consensus

ACM Reference Format:
Georgiou, Ch., Gilbert, S., Guerraoui, R., and Kowalski, D. R. 2013. Asynchronous gossip. J. ACM 60, 2, Article 11 (April 2013), 42 pages.
DOI: http://dx.doi.org/10.1145/2450142.2450147

1. INTRODUCTION

Throughout history, and much to the misfortune of humankind, epidemics have proved fast, efficient, and hard to disrupt. They have inspired, hopefully to the benefit of humankind, efficient and robust mechanisms for disseminating information in distributed systems. Such mechanisms are usually called *epidemic*, or *gossip* protocols, for they also resemble the way rumors are spread among a population.

Abstractly speaking, processes start with initial values, called *rumors*, and a gossip protocol seeks to efficiently spread those rumors among all processes. Gossip protocols have long been studied in various distributed computing contexts; see,

An extended abstract of this article appeared in the *Proceedings of ACM PODC 2008*, pp. 135–144.
The work of Ch. Georgiou was supported by research funds from the University of Cyprus. The work of S. Gilbert was supported by Singapore grant MOE2011-T2-2-042. The work of D. R. Kowalski was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/G023018/1 and EP/H018816/1].
Authors' addresses: Ch. Georgiou, Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus; email: chryssis@cs.ucy.ac.cy; S. Gilbert, Department of Computer Science, National University of Singapore, Singapore 119077; email: seth.gilbert@comp.nus.edu.sg; R. Guerraoui, School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, EPFL CH 1015 Lausanne, Switzerland; email: rachid.guerraoui@epfl.ch; D. R. Kowalski, Department of Computer Science, University of Liverpool, Liverpool L69 3BX, United Kingdom; email: D.Kowalski@liverpool.ac.uk.

DOI: http://dx.doi.org/10.1145/2450142.2450147

for example, Demers et al. [1987] (database consistency), Van Renesse et al. [1998] (failure detection), [Birman et al. 1999; Eugster et al. 2003; Gupta et al. 2002; Luo et al. 2003] (group multicast), Kermarrec et al. [2003] (group membership), Chlebus and Kowalski [2006a, 2006b] (consensus), [Georgiou et al. 2005; Georgiou and Shvartsman 2008] (load balancing), and Kempe et al. [2004] (resource location). All such protocols are essentially variants of the same simple scheme in which each process periodically sends its rumor—along with any new rumors that it has learned—to another randomly selected process. Such a scheme is quite robust due to the random pattern of communication. Two natural questions, however, arise with respect to such a scheme: how often should a process transmit its rumor, and when should a process stop?

In a synchronous system, assuming bounds on communication delays and relative process speeds, both questions are readily answered: each process sends one message per round of communication, and the processes can halt, with high probability, after a certain number of rounds. In a seminal paper, Karp et al. [2000] show indeed that, in a system of n processes, a single rumor can be disseminated in $O(\log n)$ rounds using $O(n \log\log n)$ messages, with high probability.

But how reasonable is it to assume synchrony? Gossip protocols are considered effective means to disseminate information in large scale distributed applications but is it realistic to assume that such applications are synchronous? Think of that e-mail that took two days to arrive.

While it is common to argue that distributed applications are synchronous most of the time, it is however 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. Clearly, it is appealing to devise asynchronous gossip algorithms that do not make use of any known bound on synchrony.

The simple gossip scheme sketched here 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 question remains, however, to determine when to stop gossiping; the challenge arises in part since failed processes can be confused with slow ones in the absence of synchrony bounds. Unlike in the case of a synchronous system, it is not sufficient to simply repeat the gossip step a predetermined 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.

These two questions—how often to gossip and when to stop—might appear simple, yet they are fundamental and challenging. We argue that these questions lie at the heart of determining the complexity of asynchronous gossip and hence are important for understanding when gossip is and is not effective. These questions are challenging, which might explain why theoretical work on gossip protocols focuses predominantly on synchronous systems. In fact, the very definition of the complexity of gossip in *asynchronous* systems with (potentially) infinitely increasing process relative speed and communication delays is unclear. We make the question more precise as follows: is it possible to devise an asynchronous gossip algorithm that tolerates $0 < f < n$ crash failures, yet behaves efficiently when some bounds on d and δ indeed hold? (Processes may fail by crashing at any time, permanently halting their execution.) In the parlance of Dwork et al. [1988], we are looking for asynchronous gossip algorithms with *low partially synchronous* complexity. This captures the efficiency of the algorithms in the subset of executions where synchrony bounds hold but are not known to the algorithm [Dwork et al. 1988]. However, the algorithm is indeed asynchronous and the processes have no global clocks, nor do they manipulate the synchrony bounds.

We focus in this article on two different adversarial models. In both models, an "adversary" is responsible for determining when the processes are scheduled, when processes fail, and the latency of each message. In the first model, we think of the adversary as *adaptive*, that is, it determines the schedule, failures, and message latencies in an on-line fashion, as the execution proceeds. In the second model, the adversary is *oblivious*, that is, it decides the schedule, failures, and message latencies prior to the beginning of the execution (and it cannot adapt to decisions made by the algorithm). An adaptive adversary effectively captures the worst-case performance. By contrast, an oblivious adversarial model implicitly assumes a certain amount of independence between the choices made by the algorithm and the random choices made by the algorithm. In some cases, an oblivious adversary well reflects reality, while in other cases, the choices made by the algorithm (e.g., how many messages to send) may be correlated with the choices made by the adversary (e.g., speed at which a process takes steps).

Contributions

(1) *An* adaptive *adversary* can *impose significant delays or large number of messages in asynchronous gossip.* Our first result demonstrates the inherent cost of asynchrony and crashes. Indirectly, this result indicates that the techniques from the synchronous world developed in Chlebus and Kowalski [2006a, 2006b] (for example), cannot be efficiently brought to an asynchronous environment. Specifically, we show in Theorem 3.1 (Section 3) 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, if $f = \Theta(n)$, then any protocol that improves on the trivial solution in message complexity 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 [Chlebus and Kowalski 2006b].

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.

As a corollary of our lower bound (Corollary 3.2), 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, 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

Table I. Comparing Gossip Protocols under Adaptive (Ad) or Oblivious (Ob) Adversaries, for Synchronous (Syn.) and Partially Synchronous (Part. Syn.) Models

Algorithm	Time	Messages	Model	Adv.
[Chlebus and Kowalski 2006b]	$O(\text{polylog}(n))$	$O(n \text{ polylog}(n))$	Syn.	Ad
Trivial	$O(d+\delta)$	$\Theta\left(n^2\right)$	Part. Syn.	Ad
Lower Bound (Section 3)	$\Omega\left(f(d+\delta)\right)$ or	$\Omega\left(n+f^2\right)$	Part. Syn.	Ad
EARS (Section 4)	$O(\frac{n}{n-f}\log^2 n(d+\delta))$	$O(n\log^3 n(d+\delta))$	Part. Syn.	Ob
SEARS (Section 5)	$O(\frac{n}{\varepsilon(n-f)}(d+\delta))$	$O(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)}\log n(d+\delta))$	Part. Syn.	Ob
TEARS (Section 6)	$O(d+\delta)$	$O(n^{7/4}\log^2 n)$	Part. Syn.	Ob

algorithm. When $f = \Theta(n)$, this implies a factor of $\Theta(n)$ loss either in time or message complexity.

(2) *An* oblivious *adversary* cannot *impose significant delays or large number of messages in asynchronous gossip.* We proceed to ask whether efficient asynchronous gossip is possible in the context of an *oblivious* adversary. We present three different algorithms that encompass different trade-offs between time and message complexity. The results are summarized in Table I.

The first algorithm (see Section 4), 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 $O(\frac{n}{n-f}\log^2 n(d+\delta))$ 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 Karp et al. [2000] 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 5), 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 6), 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, for $f < n/2$, asymptotically optimal *constant* time $O(d+\delta)$, with respect to n, and *strictly subquadratic* message-complexity $O(n^{7/4}\log^2 n)$, with no dependence on d or δ.

(3) *Applications to 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 Canetti and Rabin [1993], or Attiya and Welch [2004, Section 14.3]).

Table II. Consensus Protocols under an Oblivious Adversary

Algorithm	Time	Messages
Canetti-Rabin [Canetti and Rabin 1993]	$O(d+\delta)$	$O(n^2)$
CR-EARS (Sections 3,6)	$O(\log^2 n(d+\delta))$	$O(n\log^3 n(d+\delta))$
CR-SEARS (Sections 4,6)	$O(\frac{1}{\varepsilon}(d+\delta))$	$O(\frac{1}{\varepsilon}n^{1+\varepsilon}\log n(d+\delta))$
CR-TEARS (Sections 5,6)	$O(d+\delta)$	$O(n^{7/4}\log^2 n)$

(For consensus $f < n/2$ is assumed.) The resulting protocols have time and message-complexity asymptotically equal to our gossip protocols (see Section 7). The results are summarized in Table II; 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 expected constant time (with respect to 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 [1983]; 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 [1989], Aspnes [2003] and the book by Attiya and Welch [2004]. 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.

Other Related Work

As recalled earlier, in a synchronous system, a single rumor can be disseminated in $O(\log n)$ rounds using $O(n \log\log n)$ messages, with high probability [Karp et al. 2000]. One could achieve a derandomized *deterministic* synchronous protocol, 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 [Chlebus and Kowalski 2006b], even when up to $n-1$ processes may crash. (See also Chlebus and Kowalski [2006a] and Georgiou et al. [2005].) Perhaps unsurprisingly, globally synchronized gossip periods are key to obtaining such good performance.

In the context of *asynchronous networks*, Verma and Ooi [2005] 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. [2006] 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 Verma and Ooi [2005] and Boyd et al. [2006] 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 Pelc [1996] and Hromkovic et al. [2005].

Recently, Censor Hillel and Shachnai [2010] considered a variation of gossip which they call *partial information spreading*: instead of requiring each rumor to be received by all n processes, they consider a relaxed requirement where only n/c processes need to receive each rumor, and every process should receive n/c rumors, for some $c \geq 1$. The majority gossip we consider in the present work can be viewed as a special case of partial spreading (when $c \in (1,2)$). However, they consider partial spreading in a *fault-free synchronous* environment.

2. 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 Dwork et al. [1988].

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 length: if a nonfailed 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 noncrashed 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 processes that crash during each time step (subject to the restriction that throughout the entire execution, no more than f processes are crashed). We say that an execution is controlled by a (d, δ)-adversary if d and δ are the maximum delivery time and maximum step size, respectively, of that execution. 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 of the protocol up to this point).

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. Initially, the collection of rumors at process $\mathbf{p}$ holds only rumor r_p, that is, $\mathbf{p}$'s initial rumor.

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' collection, then it is the initial rumor for some process; and
(3) *Quiescence*. Eventually, every process stops sending messages forever.

These properties are required to hold, regardless of the timing properties of the system, regardless of d and δ, as long as every correct process continues to take steps and every message is eventually delivered.

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 of A 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 $M_A^{\text{asynch}}(d, \delta)$. Where it is clear from the context, we simply use M and T to abbreviate $M_A^{\text{asynch}}(d, \delta)$ and $T_A^{\text{asynch}}(d, \delta)$.

For a synchronous algorithm $\widehat{A}$, where by assumption $d = \delta = 1$ and this is known a priori by the algorithm, we define $T_{\widehat{A}}^{\text{synch}}$ and $M_{\widehat{A}}^{\text{synch}}$: for every $f < n$, for every infinite execution of $\widehat{A}$ with bounds $d = 1$ and $\delta = 1$, every correct process completes by (expected) time $T_{\widehat{A}}^{\text{synch}}$, and the (expected) number of point-to-point messages sent by all the processes combined is no more than $M_{\widehat{A}}^{\text{synch}}$.

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.

3. THE COST OF ASYNCHRONY

We now show that no randomized gossip protocol can be both time and message efficient with 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 $\Omega(n)$, or an inflation of message-complexity by a factor of $\Omega(n)$.

Underlying the lower bound lies a strategy for the adversary to fight the spreading of a rumor by adaptively choosing how to delay communication 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 prevents 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 3.1. *For every asynchronous 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) $M_A^{\text{asynch}}(d, \delta) = \Omega(n + f^2)$; or (2) $T_A^{\text{asynch}}(d, \delta) = \Omega(f(d + \delta))$.*

PROOF. Consider some asynchronous gossip algorithm A. An $\Omega(n)$ lower bound for the number of messages is straightforward as each rumor needs to be sent at least once. Thus, we show that there is either a lower bound of $\Omega(f^2)$ on the number of messages or a lower bound of $\Omega(f(d + \delta))$ on the time complexity.

We assume without loss of generality that $f \leq n/4$; otherwise, the adversary proceeds according to the same 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 allows the processes in set S_1 to run the algorithm A with $d = 1$ and $\delta = 1$ (from the perspective of processes in S_1) until every process in S_1 completes the protocol and ceases sending messages. Let t be the (global) time at which this occurs. If $t > f$, then we are done: the adversary can design an indistinguishable execution in which the processes in S_2 fail at time 0, resulting in an execution in which $d = \delta = 1$ and $t = \Omega(f(d + \delta))$. We thus assume for the remainder of this proof that $t \leq f$.

Next, consider set S_2. By choosing $\delta = f$, the adversary can delay all the processes in S_2 until time t, scheduling only processes in S_1 during this interval of time. Next, for each process $\mathbf{p} \in S_2$, the adversary precomputes the result of process $\mathbf{p}$ acting as follows: (i) receiving all the messages sent to it from S_1, and then (ii) executing $f/2$ local steps in isolation, that is, during which $\mathbf{p}$ receives no other messages from any other

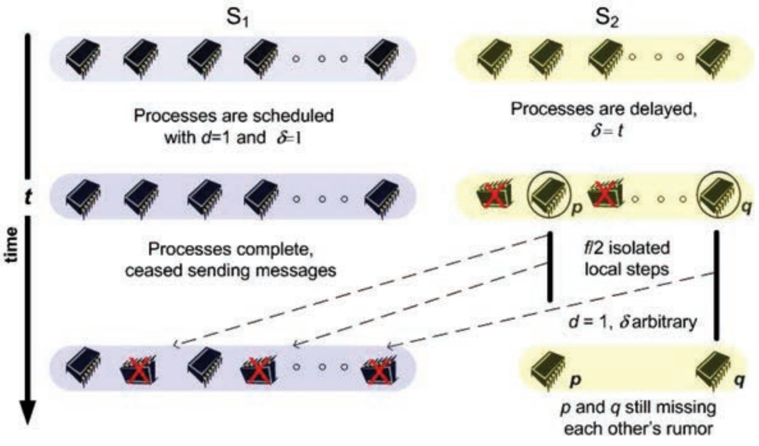
S1
S2
time
t
Processes are scheduled with d=1 and δ=1
Processes are delayed, δ= t
p
q
Processes complete, ceased sending messages
f/2 isolated local steps
d = 1, δ arbitrary
p and q still missing each other's rumor

Fig. 1. Illustration of the construction in Theorem 3.1.

process (i.e., processes in S_2). The adversary “simulates” this hypothetical scenario to determine what would happen if such a schedule were chosen.

Since the behavior of $\mathbf{p}$ is probabilistic, this “precomputation” yields a distribution over the set of messages sent by $\mathbf{p}$ during these $f/2$ steps. 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 (i.e., $|P| \geq f/4$), then we construct an execution in which $M(d,\delta) = \Omega(f^2)$. Otherwise (i.e., $|P| < f/4$), we construct an execution in which $T(d,\delta) = \Omega(f(d+\delta))$.

Case 1. $|P| \geq f/4$. Assume there are at least $f/4$ promiscuous processes. Then, after time t, the adversary schedules all of the processes in S_2 in each of the following $f/2$ time steps (i.e., $\delta = 1$ from the perspective of these processes in S_2). The adversary ensures that none of the messages sent during these steps are delivered, that is, $d \geq f/2+1$. 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. $|P| < f/4$. Assume that fewer than $f/4$ processes are promiscuous. Let $S = S_2 \setminus P$ (the set of nonpromiscuous processes), and let $\nu = |S|$. The adversary proceeds to identify two nonpromiscuous 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 nonpromiscuous processes, for each nonpromiscuous $\mathbf{p} \in S$, we define the set $N(\mathbf{p})$ to be the set of processes that $\mathbf{p}$ sends a message to with probability smaller than $1/4$ during $f/2$ (isolated) local steps. If $\mathbf{p}$ is nonpromiscuous, then the expected number of messages sent by $\mathbf{p}$ is smaller than $f/32$. Thus, $|N(\mathbf{p})| > 7f/8$: if not, then there exist (at least) $(n-7f/8) \geq f/8$ processes that $\mathbf{p}$ sends (at least) one message with probability at least $1/4$, implying that in expectation $\mathbf{p}$ sends at least $f/32$ messages, resulting in a contradiction.

Also, notice that there are at least $f/4$ nonpromiscuous processes: by definition $\nu = |S_2| - |P|$; by the way in which the partitions were chosen, $|S_2| = f/2$; by assumption, $|P| < f/4$; thus, we conclude that $\nu \geq f/4$.

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

We claim, then, that there exist two nonpromiscuous 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 nonpromiscuous 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 at least $\nu/2$ outgoing edges, there are a total of at least $\nu^2/2$ 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. Fix such a $\mathbf{p}$ and $\mathbf{q}$ for the remainder of the proof.

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, that is, $d = 1$. 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 $\mathbf{p}$ or $\mathbf{q}$ sends a message. (Notice that these processes in S_1 are currently dormant, believing that the protocol has terminated; a message from $\mathbf{p}$ or $\mathbf{q}$ might cause them to wake up, and hence the adversary fails them to prevent this.) All other processes in S_2 have already been failed, and hence no action is taken if $\mathbf{p}$ or $\mathbf{q}$ sends one of them a message.

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 $(1-(7/16+7/16)) = 1/8$. 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, $T_A^{\text{asynch}}(d,\delta) = \Omega(f(d+\delta))$, 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:

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

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

We conclude from Theorem 3.1 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 gossip algorithms.

COROLLARY 3.2 (COST OF ASYNCHRONY). *For every asynchronous gossip algorithm A, subject to an adaptive adversary, either:*

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

or

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

4. EFFICIENT EPIDEMIC GOSSIP

We have shown that in the context of an adaptive adversary, asynchronous gossip is inherently inefficient. In this section, we focus on gossip in the context of an *oblivious* adversary, and give an efficient asynchronous gossip protocol.

We begin in Section 4.1 by presenting 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 4.2 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 $O(n \log^3 n(d+\delta))$ message complexity.

4.1. 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; and (3) *Exchange*. Every so often, every pair of correct processes in the core exchange information about who has been shot. We remark that similar techniques of gathering&exchange followed by shooting&exchange were used in for example, Chlebus and Kowalski [2006a], however, they were used in a fully synchronized context where switching between specific activities was scheduled to specific rounds; in our work, these techniques needed to be appended with more adaptive mechanisms to cope with the asynchrony of the system.

Overview. At a high level, the algorithm works as follows: Whenever a process $\mathbf{p}$ is scheduled, it randomly chooses a process $\mathbf{q}$ and sends it a message containing all the rumors previously known to $\mathbf{p}$. At this point, $\mathbf{p}$ records the fact that $\mathbf{q}$ has been informed of this set of rumors. This information regarding which processes have been informed of which rumors is also attached to every message. When a process $\mathbf{p}$ discovers that every other process in the system has already been informed of every rumor that it knows about, then it enters a shut-down phase. During the shut-down phase, which continues for $\Theta(\frac{n}{n-f} \log n)$ iterations, the process $\mathbf{p}$ continues to behave in the normal fashion, processing incoming messages and sending messages to randomly chosen processes. During this shut-down phase, $\mathbf{p}$ propagates to the other processes the fact that every process has already been informed. If the process completes $\Theta(\frac{n}{n-f} \log n)$ consecutive shut-down iterations, then it becomes quiescent and stops sending messages. If at any time, either during the shut-down phase or after becoming quiescent, process $\mathbf{p}$ discovers a new rumor that some process has not been informed of, then it exits the shut-down/quiescent mode and continues as before.

Details. In more detail, the algorithm proceeds as follows. (Its 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

```
EARS($r_p$)
▷ Initialization:
$V(\mathbf{p}) \leftarrow \{r_p\}$            ▷ a set of rumors
$I(\mathbf{p}) \leftarrow \{(r_p, \mathbf{p})\}$      ▷ 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 ▷ in each scheduled step:
    ▷ First, deliver any messages received:
    for every message $m = \langle V, I \rangle$ received do
        $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \cup m.V$
        $I(\mathbf{p}) \leftarrow I(\mathbf{p}) \cup m.I$

    ▷ Second, perform computation:
    $L(\mathbf{p}) \leftarrow \{\mathbf{q} : \exists r \in V(\mathbf{p}), (r, \mathbf{q}) \notin I(\mathbf{p})\}$
    if $L(\mathbf{p}) = \emptyset$
        then  $sleep\_cnt \leftarrow sleep\_cnt + 1$
        else  $sleep\_cnt \leftarrow 0$

    ▷ Third, send messages:
    if $sleep\_cnt < \Theta\left(\frac{n}{n-f} \log n\right)$ then
        ▷ Do epidemic transmission:
        Choose q uniformly at random from $[n]$.
        Send $\langle V(\mathbf{p}), I(\mathbf{p}) \rangle$ to process q.
        for every $r \in V(\mathbf{p})$ do
          $I(\mathbf{p}) \leftarrow I(\mathbf{p}) \cup (r, \mathbf{q})$
```

Fig. 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.

informed-list $I(\mathbf{p})$ which contains pairs of rumors and processes: when $(r, \mathbf{q}) \in I(\mathbf{p})$, this implies that **p** knows that rumor r has been sent to process **q** by some process.

In each local step, a process sends a message containing $V(\mathbf{p})$ and $I(\mathbf{p})$ to a process **q** chosen uniformly at random from $[n]$. Process **p** then adds all pairs $(r, \mathbf{q})$, for $r \in V(\mathbf{p})$, to the informed-list $I(\mathbf{p})$. This implies that **p** can guarantee that **q** will eventually be informed of every rumor in $V(\mathbf{p})$. When process **q** receives a message from **p**, it updates its local sets $V(\mathbf{q})$ and $I(\mathbf{q})$, taking the union of the existing sets with the sets sent in the message.

Let $L(\mathbf{p})$ be the set of processes that **p** cannot determine (via $I(\mathbf{p})$) whether they have been sent every rumor in $V(\mathbf{p})$, that is, $L(\mathbf{p}) = \{\mathbf{q} : \exists r \in V(\mathbf{p}), (r, \mathbf{q}) \notin I(\mathbf{p})\}$. When $L(\mathbf{p})$ is empty for process **p**, then every rumor known to **p** has been sent to every process.

Notice, however, that processes may be both added and removed from $L(\mathbf{p})$ as the execution progresses. For example, initially process **p** only knows about its own rumor r_p. Consider an execution in which it does not receive any messages for a very long time. During that extended period of time, process **p** sends r_p to every other process in the system. At this point, $L(\mathbf{p}) = \emptyset$: process **p** does not know of any rumor that has not yet been propagated to everyone. However, as soon as process **p** receives a message from some process **q**, it will learn of rumor r_q. Rumor r_q, however, may not yet have been sent to all processes. At this point, process **p** will add to $L(\mathbf{p})$ every process that may not yet have received r_q.

When $L(\mathbf{p}) = \emptyset$, process $\mathbf{p}$ begins the shut-down phase. That is, if $L(\mathbf{p}) = \emptyset$, then $\mathbf{p}$ increments a counter *sleep_cnt*; otherwise, the counter *sleep_cnt* is reset to zero (effectively canceling the shut-down phase). If the *sleep_cnt* reaches $\Theta(\frac{n}{n-f}\log n)$, then process $\mathbf{p}$ becomes quiescent, that is, it ceases to send messages. Notice this only happens if there are $\Theta(\frac{n}{n-f}\log n)$ consecutive iterations in which $L(\mathbf{p}) = \emptyset$. During this period where $0 < \mathit{sleep\_cnt} < \Theta(\frac{n}{n-f}\log n)$, we say that $\mathbf{p}$ is in shut-down mode. When $\mathit{sleep\_cnt} \geq \Theta(\frac{n}{n-f}\log n)$, we say that $\mathbf{p}$ is quiescent.

During the shut-down phase, $\mathbf{p}$ continues as before, receiving messages from other processes and sending messages to randomly chosen processes. (The shut-down phase is long enough to ensure that $\mathbf{p}$ distributes its informed-list $I(\mathbf{p})$ to the other processes in the system.) If, during the shut-down phase, $\mathbf{p}$ receives a new rumor that has not yet been sent to some process, that is, if $L(\mathbf{p})$ becomes nonempty, then $\mathbf{p}$ "aborts" the shut-down phase, resetting *sleep_cnt* to zero.

In the same way, even after $\mathbf{p}$ becomes quiescent and stops sending messages, it continues to receive and process messages from other processes that are still awake. Again, if $\mathbf{p}$ receives a new rumor that has not yet been sent to some process, i.e., if $L(\mathbf{p})$ becomes nonempty, then $\mathbf{p}$ awakens and resumes the normal epidemic process until $L(\mathbf{p})$ becomes empty again, that is, *sleep_cnt* is reset to zero.

Discussion. Notice that this protocol would continue to work in the context of an adaptive adversary, however, it would be extremely inefficient. Specifically, every correct process will eventually be informed of every rumor initiated by a correct process, as a process only goes quiescent if it has evidence that its rumor has been sent to everyone. If every message is eventually delivered, then the gossip protocol still behaves correctly. Unfortunately, an adaptive adversary can induce extremely poor performance. For example, assume that the adversary targets a specific process $\mathbf{p}$ and fails every process $\mathbf{q}$ that is sent a message by $\mathbf{p}$. By doing this, the adversary can ensure that $\mathbf{p}$ has to send $\Omega(f)$ messages and the gossip protocol cannot complete for $\Omega(f(d+\delta))$ steps, where $d = 1$. Given the lower bound in Section 3, this poor performance in the face of an adaptive adversary is to be expected.

4.2. Analysis of Algorithm EARS

In this section, we analyze algorithm EARS (see Section 4.1), showing that it has time complexity $O(\frac{n}{n-f}\log^2 n(d+\delta))$ and message complexity $O(n\log^3 n(d+\delta))$, with high probability, under an oblivious adversary.

Fix a (d, δ)-adversary, and some adversarial scheduling of the epidemic gossip algorithm for n processes and $f < n$ failures. (Since the adversary is oblivious, this schedule is fixed prior to the random choices being made.)

Recall that when a process enters the shut-down phase of the protocol, it continues for $\Theta(\frac{n}{n-f}\log n)$ further steps before becoming quiescent. Assume that the hidden constant is equal to $32c$, for some constant c. Throughout the analysis, we assume that c and n are sufficiently large.

Overview. We begin with a high-level overview of the analysis. First, we divide the execution into epochs such that at most a constant fraction of the processes fail in each epoch. We argue that there is some epoch i in which: (i) there are approximately $n/2^i$ processes that have not yet failed, and (ii) each nonfailed process sends $\Theta(2^i\log^2 n)$ messages. (See Lemma 4.2 and Lemma 4.3.) We define the *core* A to be the set of approximately $n/2^i$ processes that survive epoch i. Notice that every correct process is, obviously, a member of the core.

At this point, we divide epoch i into seven stages of (approximately) equal length, that is, each process takes $\Theta(2^i \log^2 n)$ steps in each stage. Fix some process **p** that is a member of the core. We argue (in Lemma 4.4) that everything known to **p** at the beginning of a stage is successfully gossiped to every other member of the core by the end of that stage. This occurs via the standard epidemic gossip process; the only difficulty lies in coping with processes that may already have become quiescent.

From this, we conclude that at the end of the second stage, every rumor that is known to any one nonfailed process is known to every process in the core (Lemma 4.5). This follows from the fact that by the end of stage 1, every such rumor is known to at least one member of the core, and hence by the end of the second stage, it is known to all members of the core. Notice that this shows not just that every rumor initiated by a correct process is known to the core; to ensure that the shut-down process operates correctly, we need to ensure that every extant rumor (whether it was initiated at a correct or a failed process) is known to every member of the core. After the end of stage 2, no member of the core learns of a new rumor, and hence once a member of the core begins the shut-down phase, it continues to become irrevocably quiescent (as per Observation 4.6).

Next, we show that by the end of stage 3, every rumor known collectively by the core has been sent to every process in the system (see Lemma 4.7). This follows, roughly from the fact that there are $\Theta(n/2^i)$ processes each sending $\Theta(2^i \log^2 n)$ random messages, which is sufficient to ensure that every process is sent at least one message.

From this we conclude that, by the end of stage 4, at least one process has entered the shut-down phase, as during the phase, processes in the core exchange information on which messages were sent to which processes (see Lemma 4.8). Once a process **q** receives a message from a process **p** that is already in the shut-down phase, then process **q** also enters the shut-down phase, as it learns from **p** that all the rumors have been sent to all the processes. Hence, by the end of stage 5, every process in the core irrevocably enters the shut-down phase (see Lemma 4.10), and no process exits this phase in the next two stages. Hence, we conclude (see Theorem 4.11) that every process is irrevocably quiescent (or failed) by the end of stage 7. The time and message complexity bounds then follow immediately. We now proceed to present the proof in more detail.

Epochs. We begin the analysis by partitioning the execution into epochs such that in each epoch, only a constant fraction of the processes fail. Formally, we have the following definition.

Definition 4.1. Epoch 0 begins at time 0. Epoch i ends (and epoch $i + 1$ begins) at the earliest time step such that there are only $n/2^{i+1}$ non-crashed processes.

We note the following two facts regarding the epoch structure.

LEMMA 4.2. *There are at most* $\log \frac{n}{n-f}$ *epochs in an execution.*

PROOF. This follows immediately from the fact that at most f processes crash in an execution. □

LEMMA 4.3. *By time* $(7c)\frac{n}{n-f} \log^2 n(d + \delta)$*, there is some epoch* i *of length at least* $(7c)2^i \log^2 n(d + \delta)$.

PROOF. Assume this is not the case. Then, the sum of the first $\log(\frac{n}{n-f})$ epoch lengths is bounded by:

$$\sum_{i=0}^{\log \frac{n}{n-f}} (7c) \cdot 2^i \log^2 n(d+\delta) \leq (7c)\frac{n}{n-f} \log^2 n(d+\delta).$$

Thus, time $(7c)\frac{n}{n-f} \log^2 n(d+\delta)$ is part of some epoch $> \log \frac{n}{n-f}$, contradicting the fact that there are only $\log \frac{n}{n-f}$ epochs. □

Fix i to be the first epoch of length at least $(7c)2^i \log^2 n(d+\delta)$. For the remainder of the proof, we restrict our attention to this epoch. Let A be the set of processes that are non-faulty through the end of epoch i. By assumption, we have $n/2^{i+1} < |A| \leq n/2^i$. Intuitively, processes in A gather all the rumors in the system and ensure that they are sent out to all other processes before the epoch completes. By exchanging information amongst themselves, they determine when it is safe to shut-down.

Partition epoch i into 7 consecutive *stages*, each of length $c \cdot 2^i \log^2 n(d+\delta)$ (except possibly the last stage, which may be longer, as the epoch may be longer than $(7c)2^i \log^2 n(d+\delta)$). Since each process is scheduled for a local step every time δ, we can be certain that each process in A takes at least $c \cdot 2^i \log n$ local steps in each stage. We argue that by the end of the first stage, every rumor that needs to be collected is known to some process in A, and by the end of the second stage, every process in A has learned every such rumor. In the third stage, we show that every rumor has been sent to every process not in A. By the end of the fourth stage, some process has entered the shut-down phase, and in the remaining stages, every process goes to sleep.

Exchanging Information. We begin by showing 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. The basic idea is to show that the set of processes aware of a particular rumor continue to double, until a constant fraction of the processes have learned the rumor; at this point, the rumor is rapidly distributed to the remaining processes.

LEMMA 4.4 (EXCHANGE PROPERTY). *For every process $\mathbf{p} \in A$ and for every stage j of epoch i:*

(1) *All rumors known by process $\mathbf{p}$ at the beginning of stage j are known to all other processes in A at the end of stage j, with probability at least $1 - 1/n^{c/2}$.*
(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, with probability at least $1 - 1/n^{c/2}$.*

PROOF. For the purpose of showing Part (1), define $(\mathbf{p}, j)$-*data* to be the set of *rumors* known by process $\mathbf{p}$ at the beginning of stage j; for the purpose of showing Part (2), define it to be the set of *rumors* and *pairs* known by process $\mathbf{p}$ at the beginning of stage j. We estimate the probability that a given $(\mathbf{p}, j)$-data is known to a process in A at the end of stage j. (We indicate in-line where the proof for Parts (1) and (2) differ.)

First, we deal with the special case for Part (1) where $\mathbf{p}$ sleeps at some point prior to the last $d+\delta$ steps of stage j. This implies that $L(\mathbf{p}) = \emptyset$ at that point, which implies that every rumor in $V(\mathbf{p})$ has already been sent to every process. In this case, within $d+\delta$ steps of process $\mathbf{p}$ sleeping, every process has received $(\mathbf{p}, j)$-data. Assume for the remainder of the proof that $\mathbf{p}$ is awake throughout stage j, with the possible exception of the final $d+\delta$ time. (Notice that this holds by assumption for Part (2).)

We now define sets B_k, for $0 \leq k \leq \log|A|$. Each set contains processes that know $(\mathbf{p}, j)$-data, and as before, if any process in B_k is asleep (after it has learned $(\mathbf{p}, j)$-data), then we know that $(\mathbf{p}, j)$-data has already been sent to every process, and we are done (within $(d + \delta)$ time). Thus we assume that each of the processes in B_k is awake throughout stage j, with the possible exception of the final $d + \delta$ time. (Again, notice that this holds by assumption for Part (2).)

Let B_0 contain processes in A that know $(\mathbf{p}, j)$-data at the beginning of stage j. Define B_{k+1} recursively as follows, having defined sets $B_0, \ldots, B_k$: let B_{k+1} contain processes in $A \setminus (B_0 \cup \cdots \cup B_k)$ 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 $(\mathbf{p}, j)$-data. Note that each process does not sleep for at least $c \cdot \frac{n}{n-f} \log n \geq c \cdot 2^i \log n$ local steps after receiving any new rumor or pair, since it has to complete the shutdown phase before it can sleep; therefore the sets are well defined. Let $b_k = |B_0| + \cdots + |B_k|$, for $0 \leq k \leq \log|A|$. We show that the sizes of sets B_k grow at least exponentially in k, with high probability, until B_k reaches size $|A|/8$; finally, we show that $b_{\log|A|} = |A|$.

We now show that as long as $|B_k| \leq |A|/8$, then $|B_{k+1}| \geq 2|B_k|$. Notice that, trivially, $1 \leq |B_0| \leq b_0 \leq 2|B_0|$. Assume, inductively, that $2^k \leq |B_k| \leq b_k \leq 2|B_k|$, and assume that $|B_k| \leq |A|/8$. We calculate a bound on the probability that $|B_{k+1}| \leq 2|B_k|$.

Specifically, this can only be the case if there is some set S of size at most $2|B_k|$ such that every message sent by a process in $B_0 \cup \cdots \cup B_k$ is sent either to this set S *or* to another process in $B_0 \cup \cdots \cup B_k$ *or* to a process not in A. For a given set S of $2|B_k|$ processes in $A \setminus (B_0 \cup \cdots \cup B_k)$, the probability that some message is sent to a process either: in the set S or in $B_0 \cup \cdots \cup B_k$ or not in A is at most

$$1 - \frac{|A| - b_k - 2|B_k|}{n}.$$

If $|B_{k+1}| \leq 2|B_k|$, then all messages must satisfy this condition for some set S. The probability that this is true for all messages sent by processes in B_k is:

$$\begin{aligned}
&\leq \left(1 - \frac{|A| - b_k - 2|B_k|}{n}\right)^{|B_k|\cdot(c\cdot 2^i \log n - 1)}\\
&\leq \left(1 - \frac{|A|/2}{n}\right)^{|B_k|\cdot c\cdot 2^{i-1}\log n}\\
&\leq \left(1 - \frac{1}{2^{i+2}}\right)^{|B_k|\cdot c\cdot 2^{i-1}\log n}\\
&\leq e^{-(c|B_k|/8)\log n}.
\end{aligned}$$

There are at most $\binom{|A|-b_k}{2|B_k|}$ such sets S of size $2|B_k|$, and:

$$\begin{aligned}
\binom{|A| - b_k}{2|B_k|} &\leq \left(\frac{e(|A| - b_k)}{2|B_k|}\right)^{2|B_k|}\\
&\leq e^{(2|B_k|+1)\ln|A|}.
\end{aligned}$$

Taking a union bound over all such sets, we see that the probability that any set S satisfies this condition is at most $1/n^c$, implying that with high probability there is no such set S, and hence $|B_{k+1}| > 2|B_k|$. (Throughout, we assume sufficiently large n and c.) Note that for any fixed sequence of sets $B_0, \ldots, B_k$ satisfying condition $2^k \leq |B_k| \leq b_k \leq 2|B_k| \leq |A|/8$, this estimation of the conditional probability that $|B_{k+1}| \geq 2|B_k|$ holds.

When $|B_{k+1}| > 2|B_k|$, we conclude the following facts: (i) since $|B_k| \geq 2^k$, this implies that $B_{k+1} \geq 2^{k+1}$; and (ii) since $b_k \leq 2|B_k|$, this implies that $b_{k+1} = b_k + |B_{k+1}| \leq 2|B_k| + |B_{k+1}| \leq 2|B_{k+1}|$. Putting these facts together, we maintain the inductive invariant: $2^{k+1} \leq |B_{k+1}| \leq b_{k+1} \leq 2|B_{k+1}|$.

Now consider the case where $|B_k| > |A|/8$. The probability that some process $\mathbf{q} \in A \setminus (B_0 \cup \cdots \cup B_k)$ is not in B_{k+1} is at most

$$(|A| - b_k) \cdot \left(1 - \frac{1}{n}\right)^{|B_k| \cdot (c \cdot 2^i \log n - 1)} \leq |A| \cdot \left(1 - \frac{1}{n}\right)^{(n/2^{i+4}) \cdot c \cdot 2^{i-1} \log n} \leq n \cdot e^{-(c/32) \log n} \leq 1/n^c.$$

As in the previous case, this derivation holds for any fixed sequence of sets $B_0, \ldots, B_k$ satisfying condition $|B_k| > |A|/8$.

Putting the two cases together, the probability that, for every $1 \leq k \leq \log |A|$, either:

$$\begin{array}{c} 2^k \leq |B_k| \leq b_k \leq 2|B_k| \leq |A|/4 \\ \textbf{or} \\ (|B_k| > |A|/8) \quad \textbf{and} \quad (b_{k+1} = |A|) \\ \textbf{or} \\ b_k = |A| \end{array}$$

is at least

$$1 - \log |A| \cdot 1/n^c \geq 1 - 1/n^{c-1},$$

which in particular yields that $b_{\log |A|} = |A|$ with probability at least $1 - 1/n^{c-1}$. Finally note that each process in $B_0 \cup \cdots \cup B_{\log |A|}$, which is equal to A, starts having $(\mathbf{p}, j)$-data no later than time $(\log |A| \cdot c \cdot 2^i \log n - 1)(d + \delta) \leq c(2^i \log^2 n - 1)(d + \delta)$ after the beginning of stage j, that is, each process learns $(\mathbf{p}, j)$-data at some point during stage j prior to the last $d + \delta$ time.

Since there are at most $|A|$ different $(\mathbf{p}, j)$-data, the probability that each of them is known to each process in A by time $c2^i \log^2 n(d + \delta)$ after the beginning of stage j is at least $1 - |A| \cdot 1/n^{c-1} \geq 1 - 1/n^{c-2} \geq 1 - 1/n^{c/2}$. □

Gathering the Rumors. We now argue that eventually, every process in A learns every possible rumor by the end of stage 2. This will ensure that after the end of stage 2, any process that begins the shut-down phase will continue to become irrevocably quiescent.

Let $V_{\mathtt{all}}$ be the set of rumors that are eventually learned by some process in A, that is, some process that does not fail by the end of the epoch. (Notice that every correct process always "learns" its own rumor, hence $V_{\mathtt{all}}$ includes the rumor of every correct process.) When every correct process has learned $V_{\mathtt{all}}$, the gossip can safely complete. Since A contains all the correct processes, this lemma shows that the protocol successfully distributes the rumors to every correct process. (It remains afterward to bound the time and message complexity, that is, to show that eventually processes stop sending messages.) 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 4.4.

LEMMA 4.5. *At the end of stage 2, for every nonfailed process $\mathbf{p} \in A$, $V_{all} \subseteq V(\mathbf{p})$ with probability at least $(1 - 1/n^{c/4})$.*

PROOF. Fix some rumor $r \in V_{\mathtt{all}}$; we first calculate the probability that rumor r is not known to a process $\mathbf{p} \in A$ at the end of stage 1.

Notice that during any interval of length $(d + \delta)$, at least one non-failed process that knows r must be scheduled, and either (1) sleep, knowing rumor r, or (2) succeed in transmitting rumor r: otherwise, if all such processes fail, then no process knowing rumor r remains non-failed after the interval, and the rumor r is not in $V_{\texttt{all}}$. In the former case, if some sleeping process knows rumor r, then it knows that rumor r has been sent to every process; within time $d+\delta$, every process $\mathbf{q}$ in A receives rumor r and adds it to $V(\mathbf{q})$.

Consider the complementary case where no correct process that knows rumor r is asleep. During stage 1, there are at least $c \cdot 2^i \log n$ messages sent (one during each interval of $d+\delta$), each to a randomly chosen process in $[n]$. Since $|A| \geq n/2^{i+1}$, the probability that none of these messages reaches *some* process in the set A is:

$$\left(1 - \frac{|A|}{n}\right)^{c \cdot 2^i \log n} \leq \left(1 - \frac{1}{2^{i+1}}\right)^{c \cdot 2^i \log n} \leq e^{-c \log n/2} \leq 1/n^{c/2}.$$

Since rumor r is known to some process in A by the end of stage 1 with probability $(1 - 1/n^{c/2})$, and since every process in A stays non-failed until the end of phase i, by Lemma 4.4, Part (1), applied to stage 2, we conclude that rumor r is known to every process in A by the end of stage 2 with probability $(1 - 2/n^{c/2})$.

Since there are at most n rumors in $V_{\texttt{all}}$, by a union bound, we conclude that with probability $(1 - 2/n^{c/2-1}) \geq (1 - 1/n^{c/4})$, each rumor in $V_{\texttt{all}}$ is known to each process in A by the end of stage 2. □

Shut-Down and Quiescence. The key remaining portion of the analysis is to show that every process sleeps by the end of epoch i and never awakes thereafter. We begin with an observation: since, with high probability, every process in A has already learned every rumor in $V_{\texttt{all}}$ by the end of stage 2—by Lemma 4.5 and by the definition of $V_{\texttt{all}}$—it follows that no process in A learns any new rumors at any later point in the execution. As a result, if process $\mathbf{q} \notin L(\mathbf{p})$ (for some $\mathbf{p}$) after stage 2, then every rumor in $V(\mathbf{p})$ has already been sent to $\mathbf{q}$; thus, since no new rumors are added to $V(\mathbf{p})$, we can concluding the following.

Observation 4.6. For all $\mathbf{p} \in A$, no process is added back to the list $L(\mathbf{p})$ after the end of stage 2 with probability at least $1 - 1/n^{c/4}$.

We can now proceed to show that by the end of stage 3, every rumor in $V_{\texttt{all}}$ has been sent to every process in $[n]$. In particular, we show that for every process $\mathbf{q}$, there is some process $\mathbf{p} \in A$ that knows that $\mathbf{q}$ has been r-informed for every $r \in V_{\texttt{all}}$, that is, knows that $\mathbf{q}$ has been sent rumor r. This follows simply by counting the number of messages sent by (non-sleeping) processes in A, and concluding that they are sufficient to inform every process in the system.

LEMMA 4.7 (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 probability at least $1 - 1/n^{c/8}$.*

PROOF. By Lemma 4.5, we know that with probability at least $1 - 1/n^{c/4}$, every process in A knows all the rumors in $V_{\texttt{all}}$by the beginning of stage 3. Assume that this is the case. It suffices then, to show that for every $\mathbf{q}$, there exists some $\mathbf{p} \in A$ such that $\mathbf{q} \notin L(\mathbf{p})$ at some point during stage 3: by Observation 4.6, no process is added to $L(\mathbf{p})$ after the beginning of stage 3.

Also, notice that if any process $\mathbf{p} \in A$ enters the shut-down phase during stage 3, then we are done: in this case, $L(\mathbf{p}) = \emptyset$. Assume, then, that no process in A enters the shut-down phase during stage 3.

Consider some process $\mathbf{q}$ that is in every list $L(\mathbf{p})$, for $\mathbf{p} \in A$, in the beginning of stage 3. Since each process in A sends $c \cdot 2^i \log n$ messages at random during stage 3, the conditional probability that no process in A sends a message to $\mathbf{q}$ in stage 3 of epoch i is at most

$$\left(1-\frac{1}{n}\right)^{|A|\cdot c\cdot 2^i \log n} \leq \left(1-\frac{1}{n}\right)^{(c/2)\cdot n \log n} \leq e^{-(c/2)\log n} \leq 1/n^{c/2}.$$

When $\mathbf{q}$ is sent a message by some $\mathbf{p} \in A$, $(\mathbf{q}, r)$ is added to $I(\mathbf{p})$, and as a result, $\mathbf{q} \notin L(\mathbf{p})$, as required. There are at most n different processes $\mathbf{q}$, therefore the probability that some process $\mathbf{q} \in L(\mathbf{p})$ for all $\mathbf{p} \in A$ is at most $n \cdot 1/n^{c/2} = 1/n^{c/2-1}$.

Finally, we calculate the probability of failure: with probability $1/n^{c/4}$ Lemma 4.5 fails; with probability $1/n^{c/2-1}$ some process $\mathbf{q}$ is not sent a message during stage 3; thus, the probability of failure is at most $1/n^{c/8}$. □

It is therefore easy to see that by the end of stage 4, as a result of Lemma 4.7 and Lemma 4.4, at least one process has entered the shut-down phase.

LEMMA 4.8 (SINGLE SHUT-DOWN PROPERTY). *By the end of stage* 4, *at least one process* $\mathbf{p} \in A$ *has* $L(\mathbf{p}) = \emptyset$ *and has thus entered the shut-down phase with probability at least* $1 - 1/n^{c/16}$.

PROOF. Assume for the sake of contradiction that this is not the case, that is, with some probability greater than $1/n^{c/16}$ every process $\mathbf{p} \in A$ has $L(\mathbf{p}) \neq \emptyset$ at the end of stage 4.

First, notice that this holds throughout stage 4, with probability at least $(1 - 1/n^{c/4})$, by Observation 4.6. Fix some $\mathbf{p} \in A$, and assume that $\mathbf{q} \in L(\mathbf{p})$. By Lemma 4.7, we know that there exists some process $\mathbf{p}' \in A$ such that at the end of stage 3, $\mathbf{q} \notin L(\mathbf{p}')$ with probability at least $1 - 1/n^{c/8}$. By Lemma 4.4, Part (2), we know that $\mathbf{p}$ receives every pair known by process $\mathbf{p}'$ by the end of stage 4 with probability at least $1 - 1/n^{c/2}$. The union of these events occurs with probability at least $(1 - 1/n^{c/16})$, contradicting the assumption that $q \in L(\mathbf{p})$ with probability greater than $1/n^{c/16}$. □

Finally, we need to show that once one process enters the shut-down phase, soon thereafter every process enters the shut-down phase (after which all the processes sleep and the gossip algorithm completes).

We say that a process enters the shut-down phase *irrevocably* if it never exits the shut-down phase again. Notice that as soon as a process $\mathbf{p}$ in A receives a shut-down message from another process $\mathbf{q}$ in A that has already irrevocably entered the shut-down phase, process $\mathbf{q}$ enters the shut-down phase itself, as it learns that every process has been informed of every rumor (and since $\mathbf{q}$ has entered the shut-down phase irrevocably, it has already learned every relevant rumor).

Observation 4.9. If a correct process in A receives a shut-down message at time t in epoch i sent by another process in A that has irrevocably entered the shut-down phase, then it irrevocably enters the shut-down phase no later than time t.

Moreover, any process that enters the shut-down phase after stage 2 enters the shut-down phase irrevocably. We thus argue that if one process enters the shut-down phase by the end of stage 4, then every process enters the shut-down phase by the end of stage 5.

This follows from an argument similar to Lemma 4.4: Since information is rapidly exchanged among processes in A, as soon as one process enters the shut-down phase irrevocably, other processes will soon learn about this and also enter the shut-down phase irrevocably. The argument is slightly complicated by the fact that processes stop sending messages after they complete their shut-down phases. Hence, more care is needed to ensure that there are a sufficient number of shut-down messages to exchange the necessary shut-down information. (Even so, the argument is somewhat analogous to Lemma 4.4, in that we maintain increasing sized sets of processes that have entered the shut-down phase, but not yet gone to sleep.) We now conclude the following.

LEMMA 4.10 (ALL SHUT-DOWN PROPERTY). *Every process enters the shut-down phase irrevocably by the end of stage* 5 *with probability at least* $(1 - 1/n^{c/128})$.

PROOF. Let t be the time at which the first process in A enters the shut-down phase irrevocably. We show that every process in A is asleep by time $t + 2c\frac{n}{n-f}\log^2 n(d+\delta)$ with probability at least $(1 - 1/n^{c/64})$. Since, by Lemma 4.8, with probability at least $1 - 1/n^{c/16}$ some process enters the shut-down phase by the end of stage 4, and since by Observation 4.6 that process enters the shut-down irrevocably with probability at least $1 - 1/n^{c/4}$, this is sufficient to prove our claim.

Let t be the time at which the first process in A enters the shut-down phase irrevocably. We maintain a set $S \subseteq A$ of processes that have either entered the shut-down phase irrevocably or have been sent a shut-down message at a time at least t by a process that has already entered the shut-down phase irrevocably: prior to time t, the set S is empty; whenever a process in A enters the shut-down phase irrevocably, it is added to S; whenever a process in A is sent a shut-down message by a process that has irrevocably entered the shut-down phase, it is added to S.

We now define the following intervals of time, each of which is associated with a set of processes that are added to S during that interval: interval 0 begins at time t; interval k ends (and interval $k+1$ begins) at the earliest time such that either (a) 2^k processes not already associated with a previous interval have been added to S during the interval k, or (b) $|S| \geq |A|/16$ processes. In the former case, we associate the first 2^k processes added to S during interval k with interval k. (Each of the first $|A|/16$ processes added to S is associated with exactly one interval such that no interval has assigned more than 2^k processes.)

We argue that interval k ends no later than time $t + k \cdot c \cdot (n/(n-f))\log n(d+\delta)$. This claim clearly holds for interval 0 which ends at time t (when at least one process is added to S). Consider an interval k, and assume that interval k ends at time t_k. We proceed by induction to consider interval $k+1$, and argue that it completes no later than time $t' = t_k + c \cdot (n/(n-f))\log n(d+\delta)$.

We know that during interval k, 2^k new processes are added to S. Each of these 2^k processes sends $c(n/(n-f))\log n \geq c \cdot 2^i \log n$ shut-down messages by time t'. If $|S| \geq |A|/16$ by time t', then interval $k+1$ completes (by definition (b) of a interval) and the claim holds. Specifically, if $2^k \geq |A|/16$, then we can immediately conclude that $|S| \geq |A|/16$, and hence all intervals complete. Hereafter, we assume that k is such that $2^k < |A|/16$, and that $|S| < |A|/16$ at time t'.

We argue that during interval k, there are more than $2^k + 2^{k+1}$ new processes added to S, that is, enough processes to conclude interval $k+1$ (even if some of these new processes are counted in interval k). If this is not the case, then there is some set of $2^k + 2^{k+1}$ processes in $A \setminus S$ such that every shut-down message sent by a process in S during interval k is sent to one of those processes, or to a process already in S, or to a process not in A. For a given set of size $2^k + 2^{k+1}$, the probability of this is at most:

$$\left(1 - \frac{|A| - |S| - 2^k - 2^{k+1}}{n}\right)^{c\cdot 2^k \cdot 2^i \log n}$$
$$\leq \left(1 - \frac{|A|/4}{n}\right)^{c\cdot 2^k \cdot 2^i \log n}$$
$$\leq e^{-c\cdot 2^{k-3}\log n}.$$

There are at most $\binom{|A|}{2^k+2^{k+1}}$ such sets of size $2^k + 2^{k+1}$, and:

$$\binom{|A|}{2^k + 2^{k+1}} \leq \left(\frac{e|A|}{2^k + 2^{k+1}}\right)^{2^k+2^{k+1}}$$
$$\leq e^{(2^k+2^{k+1})\log|A|}.$$

We thus conclude, by a union bound, that this occurs for some set of size $2^k + 2^{k+1}$ with probability at most $1/n^{c/64}$. Since at most 2^k processes are associated with interval k, there are at least 2^{k+1} additional processes to associate with interval $k+1$, we conclude that interval $k+1$ completes by time $t_{k+1} = t'$ with probability at least $(1 - 1/n^{c/64})$, which completes the inductive claim. Since there are at most $\log n$ intervals, this holds for all intervals with probability at least $(1 - \log n/n^{c/64})$.

Finally, consider the largest interval k where $|S| < |A|/16$ at the end of interval k. We know that $k \geq \log(|A|/16) - 1$; otherwise, interval $k+1$ ends at latest when there are $|A|/32$ processes in S, contradicting our choice of k. Thus, since $k \geq \log(|A|/16) - 1$, we conclude that during interval k, there are at least $|A|/32$ new processes added to S.

Each of these processes added to S sends at least $c \cdot 2^i \log n$ shut-down messages, resulting in a total of at least $(c/64)n \log n$ (independent) shut-down messages sent during interval k. Thus with high probability, a shut-down message is sent to every process by the end of interval k. In particular, for a given process, it receives one of these shut-down messages with probability at least $(1 - 1/n^{c/64})$, and hence every process receives one of these shut-down messages with probability at least $(1 - n/n^{c/64})$.

Thus, within time $d + \delta$ past the end of interval k, each process receives a shut-down message with high probability. Since a process sleeps no later than time $c \cdot (n/(n - f)) \log n(d + \delta)$ after it receives a shut-down message (since it receives no new rumors after receiving a message from a process that has irrevocably entered the shut-down phase), we conclude that every process sleeps with high probability by time:

$$t + c \cdot \frac{n}{n-f} \log^2 n(d + \delta) + (d + \delta) + c \cdot \frac{n}{n-f} \log n(d + \delta).$$

Thus, we conclude that with probability at least $(1 - (\log n + n)/n^{c/64}) \geq (1 - 1/n^{c/128})$, every process in A is asleep by time $t + 2c\frac{n}{n-f} \log^2 n(d + \delta)$. This implies that every process enters the shut-down phase irrevocably by the end of stage 5, as desired. □

We now conclude with the main theorem.

THEOREM 4.11. *Algorithm* EARS *completes gossip with time complexity* $O(\frac{n}{n-f} \log^2 n(d + \delta))$ *and with message complexity* $O(n \log^3 n(d + \delta))$, *with high probability, subject to an oblivious adversary.*

PROOF. By Lemma 4.10, we know that every process has entered the shut-down phase by the end of stage 5 with probability at least $(1 - 1/n^{c/128})$. By an observation analogous to Observation 4.6, no process exits the shut-down phase in the next two stages with probability at least $(1 - 1/n^{c/4})$, and hence by the end of stage 7 every process has gone to sleep with probability at least $(1 - 1/n^{c/128} - 1/n^{c/4})$. This ensures that processes eventually sleep by the end of phase i.

By Lemma 4.5, we know that every rumor in $V_{\texttt{all}}$ has been learned by every process in A, and thus the gossip protocol succeeds in distributing the rumor of every correct process to every other correct process with probability at least $(1 - 1/n^{c/4})$.

Epoch i ends no later than time $O(\frac{n}{n-f} \log^2 n(d + \delta))$, 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 $n/2^k$ nonfailed 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: by the time every process sleeps at the end of stage 7, each of the at most $O(n/2^i)$ processes has sent at most $O(2^i \log^2 n(d+\delta))$ messages, resulting in $O(n \log^2 n(d+\delta))$ messages in epoch i. As we already showed, no messages are sent after stage 7 of epoch i, or in any epoch after i. Since there are at most $\log n$ epochs, the result follows. □

5. CONSTANT-TIME GOSSIP

In this section, we present a gossip algorithm, called SEARS (Spamming Epidemic Asynchronous Rumor Spreading), that achieves time complexity $O(\frac{n/\varepsilon}{n-f}(d+\delta))$, and message complexity $O(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)} \log n(d+\delta))$, for any constant $\varepsilon < 1$. The result is a constant-time gossip protocol with subquadratic message complexity, in the case where d and δ are constant, and $n - f = \Omega(n)$.

5.1. Algorithm SEARS

Algorithm SEARS is a variant of EARS in which each process sends a larger number of messages in each step. In this section, we describe SEARS.

Overview. Recall that in EARS, every time a process $\mathbf{p}$ is scheduled, it chooses one random process $\mathbf{q}$ and sends it some information (specifically the sets V and I). In the SEARS protocol, instead of choosing a single process $\mathbf{q}$, process $\mathbf{p}$ chooses a large set of processes to send messages to. This ensures that the rumors spread more quickly, and hence the protocol terminates in constant time. In addition, it is no longer necessary to perform a long shut-down phase; instead, as soon as *sleep_cnt* reaches 2, the process can enter quiescence (i.e., it stops sending messages).

Some additional care is needed to ensure that processes quiesce quickly. Recall that whenever a process learns a new rumor, it must wake up and propagate that rumor. For a rumor that is initiated at a correct process, we can show that in $O((1/\varepsilon)\frac{n}{n-f}(d+\delta))$ time, every process that is still alive has received every correct rumor. Hence, a rumor that is initiated at a correct process cannot waken a quiescent process after this point. However, a rumor that is initiated at a failing process may create problems: with some luck, the adversary may prevent such a rumor from reaching everyone sufficiently quickly, and hence it may improperly awaken quiescent processes.

To cope with this issues, we associate a counter with each rumor. The process that initiates a rumor r always associates counter value 0 with that rumor. Every other process that has received rumor r increments the counter associated with rumor r in every local step. This ensures that every time a rumor is sent by some process $\mathbf{p}$, its counter is at least one larger than the minimum value of the counter associated with rumor r that had previously been received by $\mathbf{p}$. When the counter exceeds some designated threshold τ, it is ignored and no longer awakens quiescent processes. In this way, the counter bounds the number of times a rumor is propagated, and hence prevents a failed rumor from preventing quiescence.

At the same time, the threshold is set high enough that it does not prevent the rumor from spreading to everyone. Specifically, we choose $\tau = \Theta((1/\varepsilon)\frac{n}{n-f})$. This ensures that, with high probability, the rumor of any correct process is delivered to everyone before the counter expires.

It might happen that, with some polynomially small probability, the dissemination fails to complete before the counter expires. Recall, however, that the process that initiates rumor r never increments the counter associated with r, and hence only quiesces when it is certain that every process has received r. Hence, if the process that initiated r is correct, we can be sure that in every execution, rumor r is disseminated.

In a nutshell, algorithm SEARS differs from algorithm EARS as follows.

—In each local step, each process sends messages to $\Theta(n^{\varepsilon} \log n)$ processes chosen at random.
—When *sleep_cnt* > 1, no messages are sent.
—Set V now includes pairs of rumors and counters. If a rumor's counter reaches a certain threshold τ, then this rumor is no longer disseminated.

Detailed Description. We now proceed to describe the algorithm in more detail. (Its pseudocode is given in Figure 3.) Each process $\mathbf{p}$ maintains a set of $V(\mathbf{p})$ of (rumor, counter) pairs. It also maintains a set $I(\mathbf{p})$ of informed (rumor, process) pairs. If $(\mathbf{q}, r) \in I(\mathbf{p})$, that means that $\mathbf{p}$ has *proof* that rumor r has been previously sent to process $\mathbf{q}$. Initially, $V(\mathbf{p})$ contains the pair $(r_p, 0)$, that is, process $\mathbf{p}$'s rumor with counter initialized to 0; $I(\mathbf{p})$ initially contains only (p, r_p).

Intuitively, the set $L(\mathbf{p})$ is the set of processes that have not yet been sent some nonexpired rumor, to the best of process $\mathbf{p}$'s knowledge. Formally, the set $L(\mathbf{p})$ is defined as the set of processes $\mathbf{q}$ for which there is a rumor r such that $(r, c) \in V(\mathbf{p})$ for $c < \tau$ and there is no (q, r) in $I(\mathbf{p})$. (See line 27 where $L(\mathbf{p})$ is updated.) Notice that a rumor whose counter is larger than the threshold τ is considered expired, and hence has no impact on the set $L(\mathbf{p})$. As we show later in the analysis, rumors initiated at processes that fail by time $\frac{n}{n-f}(d+\delta)$ expire by time $(1 + \tau/\frac{n}{n-f})\frac{n}{n-f}(d+\delta)$. As we choose τ to be $\Theta((1/\varepsilon)\frac{n}{n-f})$, then such rumors expire by time $O((1/\varepsilon)\frac{n}{n-f}(d+\delta))$.

As in algorithm EARS, each process maintains a counter *sleep_cnt*, initialized to 0. Once *sleep_cnt* becomes equal to 2 at a process $\mathbf{p}$, then $\mathbf{p}$ enters quiescence.

We now describe the operation of the algorithm in more detail. Conceptually, we divided each scheduled step into three parts: (i) processing incoming messages, (ii) computation, and (iii) sending gossip messages.

First, a process $\mathbf{p}$ delivers any message it has received (lines 8–19). Every nonexpired rumor received is added to $V(\mathbf{p})$ (lines 10–18). If r is a new rumor that is previously unknown to $\mathbf{p}$, then it is simply added together with the associated counter to $V(\mathbf{p})$, and the sleep counter is reset (line 18). If $\mathbf{p}$ has already received rumor r (i.e., if $(r, \cdot) \in V(\mathbf{p})$), then $\mathbf{p}$ adopts the minimum of the new counter value and its old counter value. In addition, if process $\mathbf{p}$ has already begun to quiesce, even though rumor r has not yet been spread to everyone, then in this case too, the sleep counter is reset (lines 12–13). Notice that this latter case can only occur if the counter associated with rumor r at $\mathbf{p}$ has expired, while the counter in the message received has not. Finally, regardless of what was previously stated process $\mathbf{p}$ merges the set I from the message with its own set $I(\mathbf{p})$.

Second, process $\mathbf{p}$ updates its state. It increments the counter associated with every rumor (lines 23–25), and recomputes set $L(\mathbf{p})$ (line 27). Notice that a process $\mathbf{p}$ does not increment the rumor of the process associated with its own rumor. Finally, if $L(\mathbf{p})$ is empty, that is, every process has been sent every nonexpired rumor, then $\mathbf{p}$ begins the process of quiescence, incrementing its sleep counter (lines 28–30). If process $\mathbf{p}$ increments its sleep counter in two consecutive local steps, then it stops sending messages.

Third, process $\mathbf{p}$ sends its gossip messages. If the sleep counter is at most 1, then process $\mathbf{p}$ sends $\Theta(n^{\varepsilon} \log n)$ messages to processes chosen uniformly at random from $[n]$ (lines 35–39). This gossip message contains both the set $V(\mathbf{p})$ and $I(\mathbf{p})$. At the same

```
SEARS($r_p$)
▷ Initialization:
$V(\mathbf{p}) \leftarrow \{(r_p, 0)\}$          ▷ a set of (rumors,counter) pairs
$I(\mathbf{p}) \leftarrow \{(r_p, \mathbf{p})\}$          ▷ 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  ▷ in each scheduled step:
   ▷ First, deliver any messages received:
   for every message $m = \langle V, I \rangle$ received do
         for all $(r, c) \in m.V$ where $c < \tau$ do
            if $\exists (r, c') \in V(\mathbf{p})$ and $(c < c')$ then
                  if $(sleep\_cnt > 1)$ and $(\exists\, \mathbf{q} \,:\, (r, \mathbf{q}) \notin I(\mathbf{p}))$ then
                     $sleep\_cnt \leftarrow 0$
                  $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \setminus \{(r, c')\}$
                  $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \cup \{(r, c)\}$
            elseif $\nexists (r, c') \in V(\mathbf{p})$ then
                  $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \cup \{(r, c)\}$
                  $sleep\_cnt \leftarrow 0$
         $I(\mathbf{p}) \leftarrow I(\mathbf{p}) \cup m.I$

   ▷ Second, perform computation:
   ▷ Increment rumor counters:
   for all $(r, c) \in V$ where $r \neq r_p$ do
         $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \setminus \{(r, c)\}$
         $V(\mathbf{p}) \leftarrow V(\mathbf{p}) \cup \{(r, c + 1)\}$
   ▷ Compute $L$ and update the sleep counter:
   $L(\mathbf{p}) \leftarrow \{\mathbf{q} : \exists (r, c) \in V(\mathbf{p}), c < \tau, (r, \mathbf{q}) \notin I(\mathbf{p})\}$
   if $L(\mathbf{p}) = \emptyset$
         then    $sleep\_cnt \leftarrow sleep\_cnt + 1$
         else    $sleep\_cnt \leftarrow 0$

   ▷ Third, send messages:
   if $sleep\_cnt \leq 1$ then
         ▷ Do epidemic transmission to many processes:
         repeat $kn^{\varepsilon} \log n$ times:
            Choose $\mathbf{q}$ uniformly at random from $[n]$.
            Send $\langle V(\mathbf{p}), I(\mathbf{p}) \rangle$ to process $\mathbf{q}$.
            for every $(r, c) \in V(\mathbf{p})$ where $c \leq \tau$ do
                  $I(\mathbf{p}) \leftarrow I(\mathbf{p}) \cup (r, \mathbf{q})$
```

Fig. 3. The Epidemic-style gossip algorithm SEARS, stated for process $\mathbf{p}$; r_p denotes the rumor of $\mathbf{p}$. Every time $\mathbf{p}$ is scheduled to take a step, it executes one iteration of the main loop. Suitable threshold $\tau = \Theta((1/\varepsilon)\frac{n}{n-f})$ and constant $k > 0$ are set as in the analysis, c.f., Section 5.2.

time, it records in its set $I(\mathbf{p})$ that each of these rumors was sent to the designated processes.

5.2. Analysis of Algorithm SEARS

In this section, we analyze the performance of SEARS. We begin with an overview of the analysis.

Overview. For the purpose of this overview, consider some particular rumor r that is initiated by some correct process $\mathbf{p}$. The analysis of SEARS can be divided into three parts.

In the first part, we show that rumor r reaches a constant fraction of the processes within $O(\frac{n}{n-f}(1/\varepsilon)(d+\delta))$ time. Observe that within $O(\frac{n}{n-f}(d+\delta))$ time, process $\mathbf{p}$ has sent its rumor to $\Theta(n^{\varepsilon})$ correct processes, with high probability (or all correct processes, if $\Theta(n^{\varepsilon}) > n-f$). 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} until, after no more than $\frac{n}{n-f}(1/\varepsilon)$ steps, a constant fraction of the correct processes know r, with high probability. This follows from a straightforward analysis of the epidemic spreading. At this point, 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. This follows from the fact that, at this point, we have a large number of processes each sending a large number of messages containing r. Notice that the threshold τ is sufficiently large so that, during this procedure, no process will quiesce due to the counter expiring; if any process that knows r increments its sleep counter, it is only because every process has already been sent r. This part of the analysis is described in Lemma 5.3.

The second part of the analysis shows that within a further $O(\frac{n}{n-f}(1/\varepsilon)(d+\delta))$ time, every process knows that rumor r was sent to every other process. This is a sufficient condition to allow processes to increment their sleep counter and quiesce. The analysis now is much like in the first part: the fact that rumor r has been sent to every process spreads epidemically through the system. Unlike in the first part of the analysis, processes are becoming quiescent during this procedure, and some care is needed to ensure that, despite this, the epidemic spreading succeeds. This part of the analysis is described in Lemma 5.4.

The third part of the analysis shows that every correct process enters quiescence. We have already shown that good rumors cannot prevent quiescence. We also show in Lemma 5.2 that "bad" rumors (rumors initiated by processes that fail within the first $\frac{n}{n-f}(d+\delta)$ time) cannot prevent quiescence, as their counters expire. Putting these facts together yields the final theorem. The final time and message complexities follow immediately. We now present the analysis in more detail.

Balls-in-Bins Fact. We begin with a simple fact regarding balls-and-bins that is useful in the analysis. Imagine that you have some set of n bins, some $n-f$ of which are "marked" red. (These represent the correct processes.) These are the bins we are trying to hit. Assume we are throwing some $\Theta(\frac{n}{n-f} m \log n)$ balls, uniformly at random, into these bins, where $m \leq (n-f)/2$. These balls represent the messages containing a rumor. We show that, with high probability, at least m of the red bins receive at least one ball. This allows us to relate the number of correct processes that receive a rumor to the number of messages sent, and follows by straightforward (and standard) calculation.

FACT 5.1. *Assume you have $16c\frac{n}{n-f}m\log n$ balls thrown uniformly at random into n bins, out of which $n-f$ are red, where c is a sufficiently large constant and $1 \leq m \leq$*

$(n-f)/2$. Then the probability that fewer than m red bins have at least one ball is at most $1/n^c$.

Proof. We first argue that at least $8m\log n$ balls land in red bins, with high probability. Notice that each ball independently lands in a red bin with probability $(n-f)/n$, and hence in expectation, there are $16cm\log n$ balls that land in red bins. Thus, if we consider the Chernoff bound where for any constant $0 < \zeta < 1$, $\Pr(X < (1-\zeta)16cm\log n) < e^{-(16cm\log n)\zeta^2/2}$, and set $\zeta = 1/2$, we conclude that the probability that there are fewer than $8cm\log n$ balls in red bins is $< e^{-16cm\log n/8} < 1/n^{2c}$.

Let ρ be the number of red bins that receive at least one ball. Assuming that there are at least $8cm\log n$ balls that land in red bins, we calculate the probability that $\rho \leq m$.

$$\begin{aligned}
\Pr(\rho \leq m) &\leq \left(\frac{m}{n-f}\right)^{8cm\log n}\binom{n-f}{m} \\
&\leq e^m\left(\frac{m}{n-f}\right)^{8cm\log n - m} \\
&\leq e^m\left(\frac{1}{2}\right)^{4cm\log(n)} \\
&\leq \left(\frac{1}{2}\right)^{2cm\log(n)} \\
&\leq 1/n^{2c}.
\end{aligned}$$

Taking a union bound over the two randomized claims, the result follows. □

Terminology. For the remainder of the proof, fix a (d, δ)-adversary, and some adversarial scheduling of the algorithm for n processes and f failures. For the purpose of the analysis we divide the execution into *stages*, each of length $\frac{n}{n-f}(d+\delta)$. We say that a rumor r_p is *good* if the process $\mathbf{p}$ that begin with rumor r_p does not fail prior to the end of the first stage. Every rumor that is not good is *bad*.

Bad Rumors. We begin the analysis by dispensing with the bad rumors. Since the initiating process of a bad rumor fails by the end of the first stage, we know that from that point on, the counter associated with a bad rumor is strictly increasing. Hence eventually, no process $\mathbf{q}$ adds a bad rumor to its set $V(\mathbf{q})$, and the bad rumors cease to delay quiescence.

Lemma 5.2. *Let r be a bad rumor. After the end of stage $1 + \tau/\frac{n}{n-f}$: if $(r, c) \in V(\mathbf{p})$ for some process $\mathbf{p}$, then $c \geq \tau$.*

Proof. We proceed by induction over intervals of time of length $(d+\delta)$. Consider some time t after the end of the first stage. Assume that at time t, counter value c is the minimum value of any counter associated with rumor r for all nonfailed processes and for all messages in-transit.

Observe that any process taking a step between time t and $t+(d+\delta)$ receives only messages containing counter values at least c associated with rumor r. Moreover, observe that every process taking a step between time t and $t+(d+\delta)$ increments the counter associated with rumor r: the only process $\mathbf{q}$ that does not increment the counter associated with rumor r is the process $\mathbf{q}$ that began with rumor r initially; however since rumor r is a bad rumor, this process $\mathbf{q}$ has failed prior to time t. Similarly, every message that is sent between time t and $t+(d+\delta)$, if it contains rumor r, has a counter value of at least $c+1$, as messages are sent after the counter is incremented.

Finally, notice that every process takes at least one step between time t and time $t+(d+\delta)$, and every message in-transit is delivered between time t and $t+(d+\delta)$. Thus, we conclude that by time $t+(d+\delta)$, $c+1$ is the minimum value of any counter associated with rumor r for all nonfailed processes and for all messages in-transit. Hence, by time $\frac{n}{n-f}(d+\delta)+\tau(d+\delta)$, we conclude that $c \geq \tau$. From this, the claim of the lemma follows immediately. □

From this lemma it follows that if τ is appropriately chosen to be $\Theta((1/\varepsilon)\frac{n}{n-f})$, then bad rumors expire by the end of stage $2/\varepsilon$. (We will need this property later in the analysis, c.f., Corollary 5.5.) More specifically, we choose τ such that $1+\tau/\frac{n}{n-f} = 2/\varepsilon$. Note that $\tau > (1/\varepsilon)\frac{n}{n-f}$.

Spreading the Rumors. Consider a process $\mathbf{p}$. Let $r = r_p$ be the rumor of $\mathbf{p}$. Assume that r is a good rumor, that is, $\mathbf{p}$ does not fail before the end of the first stage. The first lemma states that by the end of stage $\frac{1}{\varepsilon}$, a message containing rumor r has been sent to every process, with high probability. This lemma proceeds by showing that in every stage, the number of processes that know r increases by a factor of n^{ε}, up until a constant fraction of the processes know r. At that point, the rumor is quickly spread to everyone in the following stage.

LEMMA 5.3. *Assume* $\mathbf{p}$ *is a correct process at the end of stage 1 and* $r = r_p$ *is the rumor initiated by* $\mathbf{p}$. *For every process* $\mathbf{q} \in [n]$*: with high probability, by the end of stage* $\frac{1}{\varepsilon}$*, a message containing rumor* r *has been sent to* $\mathbf{q}$*, where the message contains rumor* r *with associated counter* $c < \tau$. *Moreover, at some point prior to the end of stage* $\frac{1}{\varepsilon}$*, there is some correct process* $\mathbf{w}$ *with* $(r, \mathbf{q}) \in I(\mathbf{w})$ *and* $\mathbf{w}$ *has counter* $c \leq (1/\varepsilon)\frac{n}{n-f}$ *associated with rumor* r.

PROOF. Consider rumor $r = r_{\mathbf{p}}$ initiated by a process $\mathbf{p}$ that is correct at the end of stage 1, and any process $\mathbf{q}$. We begin by defining some terminology. Define process $\mathbf{p}$ to be the one and only stage 0 process. Fix a constant k such that in each step, a process sends $kn^{\varepsilon}\log n$ messages (see line 35 in Figure 3).

We say that the first $k\frac{n}{n-f}n^{\varepsilon}\log n$ messages sent by process $\mathbf{p}$ are *stage 1* messages. Notice that all stage 1 messages are sent and received by the end of stage 1, since process $\mathbf{p}$ takes at least $\frac{n}{n-f}$ local steps in stage 1, and hence sends at least $k\frac{n}{n-f}n^{\varepsilon}\log n$ messages during the first stage (since it sends $kn^{\varepsilon}\log n$ messages at each local step, and by assumption, $\mathbf{p}$ does not fail prior to the end of stage 1). We say that all correct processes that receive a stage 1 message are *stage 1 processes*. (Note that some messages that are sent in stage 1 may not be stage 1 messages.)

We inductively define *stage* j messages and stage j processes. Let $\mathbf{z}$ be a (correct) stage $j-1$ process, that is, a correct process that receives a stage $j-1$ message (which contains rumor r). Then the next $k\frac{n}{n-f}n^{\varepsilon}\log n$ messages sent by process $\mathbf{z}$ after receiving its *first* stage $j-1$ message are called *stage* j messages. We say that every correct process that receives a stage j message is a stage j process. Observe that every stage j message is sent and also received (as long as the receiver is not faulty) by the end of stage j (since each stage includes at least $\frac{n}{n-f}$ local steps, and every stage $j-1$ message sent to a correct process is received by the beginning of stage j). Note that by definition, every stage j process is correct.

As an aside, observe that a given message may be a stage j message for more than one value of j, and a process may be a stage j process for more than one value of j. (For example, consider a process that receives both a stage 3 and a stage 5 message in the same local step.) Thus, different stages are not necessarily independent. All the messages sent within a given stage, however, are independent.

Notice that every stage j message has rumor r associated with a counter value $c \leq j \cdot \frac{n}{n-f}$. This follows by a simple induction argument: It holds immediately for all stage 1 messages, as the counter is only incremented once per local step (in which $kn^{\varepsilon} \log n$ messages are sent). If process $\mathbf{z}$ is a stage $j-1$ process, then it receives rumor r with counter at most $(j-1)\frac{n}{n-f}$, by induction, and hence during the following $\frac{n}{n-f}$ steps, the counter remains at most $j \cdot \frac{n}{n-f}$, as all of the stage j messages sent by process $\mathbf{z}$ are sent in the following $\frac{n}{n-f}$ local steps.

From this, we conclude that if a process $\mathbf{z}$ receives a stage j messages for $j \leq 1/\varepsilon$, then process $\mathbf{z}$ adds rumor r accompanied by the counter to set $V(\mathbf{z})$ by the end of stage j. This conclusion follows from the fact that $\tau > (1/\varepsilon)\frac{n}{n-f}$, and that a rumor is added as long as the counter does not exceed τ.

We next verify the number of stage j messages that a stage $j-1$ process sends. If a process $\mathbf{z}$ is not quiescent, that is, it has not incremented its sleep counter, then it sends $kn^{\varepsilon} \log n$ messages in each step. On the other hand, a quiescent process may send no further messages. Assume process $\mathbf{z}$ is a stage $j-1$ process for some $j \leq 1/\varepsilon$, and assume $\mathbf{z}$ does not send $k\frac{n}{n-f}n^{\varepsilon} \log n$ stage j messages. In this case, it must have incremented its sleep counter. However, we know that upon receiving its stage $j-1$ message, process $\mathbf{z}$ has a counter $c \leq (j-1)\frac{n}{n-f} < \tau$ associated with rumor r. Thus, it must be the case that $L(\mathbf{z}) = \emptyset$, and we conclude that rumor r has been sent to every process in $[n]$, from which the lemma follows. (Notice that $\mathbf{z}$ is correct, since it is a stage j process, and it has $(r, \mathbf{q}) \in I(\mathbf{z})$ for every process $\mathbf{q}$.) For the remainder of the proof, we assume that every stage j process sends all $\Theta(\frac{n}{n-f}n^{\varepsilon} \log n)$ of its stage $j+1$ messages.

We now show that for every process $\mathbf{q} \in [n]$: with high probability, a stage j message is sent to $\mathbf{q}$ for some $j \leq 1/\varepsilon$. Thus, every correct process $\mathbf{q}$ is a stage j process for some $j \leq 1/\varepsilon$. We proceed by arguing that the number of stage j processes grows by a factor of n^{ε} with each increasing stage, until there are sufficiently many stage $j+1$ messages to ensure that rumor r is sent to all n processes, with high probability.

We analyze the number of stage j processes, for $j \geq 1$, as follows. Assume, inductively, that there are at least $n^{(j-1)\varepsilon}$ stage $j-1$ processes. (Note that this is trivially true when $j = 1$.) As already discussed above, each stage $j-1$ process sends $k\frac{n}{n-f}n^{\varepsilon} \log n$ stage j messages (as otherwise we are done), that is, there are a total of $\Omega(k\frac{n}{n-f}n^{j\varepsilon} \log n)$ stage j messages. In this case, each of these stage j messages is sent to a process chosen independently and uniformly at random. There are the following two cases to consider:

(1) First, assume that $n^{j\varepsilon} > \frac{n-f}{2}$. In this case, for an appropriate choice of k, there are $\Theta(n \log n)$ stage j messages sent to randomly chosen processes. The probability that some process is not hit by one of these messages is $(1 - 1/n)^{\Theta(n \log n)} \leq e^{-\Theta(\log n)}$, that is, every process is sent a message containing rumor r with high probability.
(2) We now consider the case where $n^{j\varepsilon} \leq \frac{n-f}{2}$. By Fact 5.1, for a sufficiently large k, we conclude that at least $n^{j\varepsilon}$ correct processes receive stage j messages with high probability. That is, there are at least $n^{j\varepsilon}$ stage j processes, with high probability.

This inductive argument, taking a union bound over all the stages, shows that for all stages where $n^{j\varepsilon} \leq (n-f)/2$, there are $n^{j\varepsilon}$ stage j processes; it also implies that if j is the smallest stage where $n^{j\varepsilon} > (n-f)/2$, then with high probability, every process (whether correct or faulty) is sent a stage j message. It is easy to see that this last stage j is no later than stage $1/\varepsilon$, since $n^{\varepsilon \frac{1}{\varepsilon}} = n > (n-f)/2$.

Finally, since a stage j process is correct (by definition), this implies that for every process $\mathbf{q}$, there is some correct process $\mathbf{w}$ that has $(r, \mathbf{q}) \in I(\mathbf{w})$. $\square$

From Lemma 5.3, we can conclude that by the end of stage $1/\varepsilon$, every correct rumor has been sent to every process. For processes to quiesce, however, we also have to show that every process knows that every process has been sent every rumor. This is complicated by the fact that processes, after quiescing, stop sending messages and hence stop propagating the necessary information. In Lemma 5.4, we show that by the end of stage $2/\varepsilon$, every process has received messages informing it that every other process has received every rumor. The analysis is quite similar to Lemma 5.3, with a few additional details to cope with the ongoing quiescence.

LEMMA 5.4. *Assume* $\mathbf{p}$ *is a correct process at the end of stage* 1 *and* $r = r_p$ *is the rumor initiated by* $\mathbf{p}$. *For every process* $\mathbf{q} \in [n]$: *with high probability, by the end of stage* $\frac{2}{\varepsilon}$, *for every process* $\mathbf{u} \in [n]$ *a message has been sent to* $\mathbf{u}$ *from some correct process* $\mathbf{w}$, *where* $(r, \mathbf{q}) \in I(\mathbf{w})$.

PROOF. Consider rumor $r = r_{\mathbf{p}}$ initiated by a process $\mathbf{p}$ that is correct at the end of stage 1, and any process $\mathbf{q}$. Recall we have already shown in Lemma 5.3 that at some point prior to the end of stage $\frac{1}{\varepsilon}$, there is some correct process $\mathbf{z}$ with $(r, \mathbf{q}) \in I(\mathbf{z})$ and $\mathbf{z}$ has counter $c \leq (1/\varepsilon)\frac{n}{n-f}$ associated with rumor r. Fix $\mathbf{z}$ to be this process for the remainder of the proof. Fix a constant k such that in each step, a process sends $kn^{\varepsilon} \log n$ messages (see line 35 in Figure 3).

We now define some terminology. We define $\mathbf{z}$ to be the one and only stage 0 process. We say that the first $k\frac{n}{n-f}n^{\varepsilon} \log n$ messages sent by process $\mathbf{z}$ immediately after adding $(r, \mathbf{q})$ to $I(\mathbf{z})$ are *stage 1* messages. We say that all correct processes that receive a stage 1 message are *stage 1 processes*.

We inductively define *stage* j messages and stage j processes. Let $\mathbf{u}$ be a stage $j-1$ process, that is, a correct process that receives a stage $j-1$ message. There are two cases to consider. First, if $\mathbf{u}$ has sleep counter less than 2 after receiving and processing the stage $j-1$ message, then we designate the next $k\frac{n}{n-f}n^{\varepsilon} \log n$ messages sent by process $\mathbf{u}$ after receiving its *first* stage $j-1$ message as *stage* j messages. The second case occurs when $\mathbf{u}$ has already incremented its sleep counter past 1 and does not reset it upon receiving the stage $j-1$ message; in this case we designate the $k\frac{n}{n-f}n^{\varepsilon} \log n$ messages sent by process $\mathbf{u}$ in the local step in which it incremented its sleep counter as *stage* j messages. In either case, we say that every correct process that receives a stage j message is a stage j process.

Observe that every stage j process, upon receiving and process a stage j message, has associated with rumor r counter value $c \leq (1/\varepsilon)\frac{n}{n-f} + j \cdot \frac{n}{n-f}$, as $\mathbf{z}$ initially has counter at most $(1/\varepsilon)\frac{n}{n-f}$ immediately after adding $(r, \mathbf{q})$ to $I(\mathbf{z})$.

We now argue that every stage j message contains the fact that $\mathbf{q}$ has been sent the rumor r. This can be seen as follows: Assume inductively that a stage $j-1$ process receives a stage $j-1$ message that contains the fact that $\mathbf{q}$ has been sent rumor r. In the first case where the process has not yet incremented its sleep counter, then it follows immediately that all future messages, including the stage j messages that it sends, include the information that $\mathbf{q}$ has been sent rumor r. In the second case, consider the messages sent when the sleep counter was incremented. If, at the time, the process had not already received rumor r, then it would have reset the sleep counter (and reawakened) on (or prior to) receiving the stage $j-1$ message (see line 18). Since this did not happen, we conclude that the process knew rumor r when it incremented its sleep counter. Moreover, it only incremented its sleep counter because the set L

was empty, that is, either: (i) the counter associated with r had passed the threshold or (ii) it knew that the rumor r had been sent to every process in $[n]$ (including $\mathbf{q}$). If only condition (i) was true, then again the process would have reset its sleep counter on or prior to receiving the stage $j-1$ message (see lines 12–13). Hence, we conclude that when the process incremented its sleep counter, it already knew that rumor r had been send to process $\mathbf{q}$, and the stage j messages that it sent on incrementing its sleep counter included this fact.

We now analyze the number of stage j processes for $j \geq 1$. Assume, inductively, that there are at least $n^{(j-1)\varepsilon}$ stage $j-1$ processes, and hence $\Omega(k\frac{n}{n-f}n^{j\varepsilon}\log n)$ stage j messages. (Notice that this is trivially true for $j=1$.) Each of these stage j messages is sent to a process chosen independently and uniformly at random. There are two cases to consider.

(1) First, assume that $n^{j\varepsilon} > \frac{n-f}{2}$. In this case, for an appropriate choice of k, there are $\Theta(n \log n)$ stage j messages sent to randomly chosen processes. The probability that some process is *not* hit by one of these messages is $(1-1/n)^{\Theta(n\log n)} \leq e^{-\Theta(\log n)}$, i.e., every process is sent a stage j message, with high probability.
(2) We now consider the case where $n^{j\varepsilon} \leq \frac{n-f}{2}$. By Fact 5.1, for sufficiently large k, we conclude that at least $n^{j\varepsilon}$ correct processes receive stage j messages with high probability. That is, there are at least $n^{j\varepsilon}$ stage j processes, with high probability.

This inductive argument, taking a union bound over all the stages, shows that for all stages where $n^{j\varepsilon} \leq (n-f)/2$, there are $n^{j\varepsilon}$ stage j processes; it also implies that if j is the smallest stage where $n^{j\varepsilon} > (n-f)/2$, then with high probability, every process (whether correct or faulty) is sent a stage j message containing the fact that process $\mathbf{q}$ has already been sent rumor r (with high probability). Since every stage $1/\varepsilon$ message is received by the end of stage $2/\varepsilon$, we conclude that the lemma holds. □

Quiescence. It remains to argue that every correct process reaches quiescence. By the end of stage $2/\varepsilon$, we have shown that no correct process learns a new rumor, and hence no correct process adds a process to its list L.

COROLLARY 5.5. *For all processes* $\mathbf{p}$ *that are correct at the end of stage* $2/\varepsilon$*, list* $L(\mathbf{p}) = \emptyset$ *after the end of stage* $\frac{2}{\varepsilon}$*, with high probability.*

PROOF. Consider a process $\mathbf{p}$ that is correct at the end of stage $2/\varepsilon$. By definition, process $\mathbf{q}$ is in list $L(\mathbf{p})$ if there exists some pair $(r, c) \in V(\mathbf{p})$ where $c < \tau$ and $(r, \mathbf{q}) \notin I(\mathbf{p})$. Assume, first, that r is a bad rumor. Then, by Lemma 5.2, we conclude that if $(r, c) \in V(\mathbf{p})$, then $c \geq \tau$. Assume, then, that r is a good rumor. Then, by Lemma 5.4, we know that $\mathbf{p}$ was sent a message by the end of stage $2/\varepsilon$ where $(r, \mathbf{q}) \in I$. Thus, we know that $(r, \mathbf{q}) \in I(\mathbf{p})$. We conclude, then, that process $\mathbf{q}$ cannot be in list $L(\mathbf{p})$ after the end of stage $2/\varepsilon$. □

Finally, we conclude that every process is quiescent after stage $2/\varepsilon + 1$.

LEMMA 5.6. *By the end of stage* $\frac{2}{\varepsilon} + 1$*, all processes are quiescent, with high probability.*

PROOF. The result follows from Corollary 5.5, and the quiescence condition in lines 27–30. □

We now conclude with the main theorem.

THEOREM 5.7. *Algorithm* SEARS *solves the gossip problem with time complexity* $O(\frac{1}{\varepsilon}(\frac{n}{n-f})(d+\delta))$ *and message complexity* $O(\frac{n^{1+\varepsilon}}{\varepsilon}(\frac{n}{n-f})\log n(d+\delta))$*, with high probability under an oblivious adversary.*

PROOF. We first argue that, with probability 1, every correct rumor is eventually delivered to every correct process. This conclusion follows from the fact that the initiator of rumor r never increments the counter associated with rumor r. Hence, the initiator of rumor r, if it is correct, only quiesces when: for every process $\mathbf{q} \in [n]$, $(r, \mathbf{q}) \in I$. That is, it only quiesces when there is a proof that rumor r was sent to every process.

Next, we argue that every correct process eventually quiesces, with probability 1. When combined with the previous observation, this implies that every correct rumor is eventually delivered to every correct process. Assume for the sake of contradiction that $\mathbf{q}$ is a correct process that never quiesces. In every step, $\mathbf{q}$ sends a message to a set of randomly chosen processes. Thus, eventually, with probability 1, $\mathbf{q}$ sends a message to every process in the system. At this point, it is either the case that $\mathbf{q}$ quiesces, or $\mathbf{q}$ learns of a new rumor to propagate. This can happen at most n times, and hence eventually $\mathbf{q}$ quiesces, with probability 1.

The time complexity follows from Lemma 5.6 and the fact that each stage takes $\frac{n}{n-f}\cdot(d+\delta)$ time. The message complexity follows from the fact that in each stage, each process sends at most $O(\frac{n}{n-f}n^{\varepsilon}\log n(d+\delta))$ messages, and there are n processes. □

6. 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 [Canetti and Rabin 1993] and Do-All [Chlebus et al. 2002; Georgiou and Shvartsman 2008], 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, that is, it is *strictly* sub-quadratic. In order for majority gossip to be feasible, we need to assume that $f < n/2$; otherwise, it is clearly impossible to receive more than $n/2$ rumors.

6.1. Algorithm TEARS

We begin with on overview of the algorithm, and proceed to describe TEARS in more detail.

Overview. The TEARS protocol consists of a two-hop dissemination. In the first stage, each process initially sends its rumor to approximately $\Theta(\sqrt{n}\log n)$ randomly chosen processes. We refer to these as *first-level* messages, since they directly propagate a rumor from its initiator to a selection of other processes.

In the second stage, every so often, each process upon receiving a sufficient number of first-level messages, sends a set of second-level messages, forwarding all the rumors it has received to approximately $\Theta(\sqrt{n}\log n)$ randomly chosen processes.

The key to the TEARS algorithm is determining how often a process should send second-level messages. If it sends second-level messages too often, then the message complexity will grow too high. On the other hand, if it does not send enough second-level messages, then it is possible that not enough rumors will be propagated.

There are two rules used to determine when to send second-level messages. Throughout, a process counts the number of first-level messages it has received. When it has

```
TEARS($r_p$)
1  ▷ Initialization:
2  $a \leftarrow 4\sqrt{n}\log n$
3  $\mu \leftarrow 2\sqrt{n}\log n$
4  $\kappa \leftarrow 32\sqrt[4]{n}\log n$
5  $V(\mathbf{p}) \leftarrow \{r_p\}$
6  $\Pi_1(\mathbf{p}) : \forall \mathbf{q} \neq \mathbf{p}$ put $\mathbf{q}$ into $\Pi_1(\mathbf{p})$ with probability $a/n$
7  $\Pi_2(\mathbf{p}) : \forall \mathbf{q} \neq \mathbf{p}$ put $\mathbf{q}$ into $\Pi_2(\mathbf{p})$ with probability $a/n$
8  $flag \leftarrow$ true
9  $up\_msg\_cnt \leftarrow 0$
10
11 repeat
12     ▷ Two-Hop Transmission
13     for every process $\mathbf{q} \in \Pi_1(\mathbf{p})$ do
14             send $m = \langle V(\mathbf{p}), flag \rangle$ to $\mathbf{q}$
15     $flag \leftarrow$ false
16     for every message $m$ received do
17             $V(\mathbf{p}) = V(\mathbf{p}) \cup m.V$
18             if $m.flag =$ true
19                then $up\_msg\_cnt \leftarrow up\_msg\_cnt + 1$
20             $bcast \leftarrow$ false
21             if $(\mu - \kappa \leq up\_msg\_cnt < \mu + \kappa)$
22                then $bcast \leftarrow$ true
23             if $(up\_msg\_cnt - \mu)/\kappa$ is a positive integer
24                then $bcast \leftarrow$ true
25             if $bcast =$ true
26                then for every process $\mathbf{q} \in \Pi_2(\mathbf{p})$ do
27                        send $m = \langle V(\mathbf{p}), flag \rangle$ to $\mathbf{q}$
```

Fig. 4. Two-hop majority gossip algorithm TEARS, stated for process $\mathbf{p}$; r_p denotes the rumor of $\mathbf{p}$.

received at least $\Theta((n^{1/2} - n^{1/4})\log n)$ first-level messages, it sends a set of second-level messages *every time* it receives a new first-level message, up until it has received $\Theta((n^{1/2} + n^{1/4})\log n)$ first-level messages. This ensures that a large chunk of rumors are all forwarded.

Second, from that point on, every time it receives a further $\Theta(n^{1/4}\log n)$ new first-level messages, it again sends a set of second-level messages. This ensures that if a large number of rumors trickle in late, they also get forwarded.

Together, this two-stage dissemination ensures that every process receives a majority of the rumors, with high probability.

Details. We describe algorithm TEARS from the point of view of a process $\mathbf{p}$. (Its pseudocode is presented in Figure 4.) We define three additional parameters to simplify the description of the algorithm: $a = 4\sqrt{n}\log n$, $\mu = \frac{a}{2}$, and $\kappa = 32n^{1/4}\log n$.[1] 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 $\Pi_2(\mathbf{p})$, respectively) with probability a/n, independently at random (lines 6–7).

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

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 $\Pi_1(\mathbf{p})$ (lines 13–14). Recall that we call such messages *first-level messages*. 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 $\Pi_2(\mathbf{p})$ (lines 21–22). It does the same after receiving $\mu + j$ first-level messages, for every $-\kappa < j < \kappa$ (lines 21–22), and later after receiving $\mu + i\kappa$ first-level messages, for every positive integer i (lines 23–24).

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.

6.2. Analysis of Algorithm TEARS

In this section, we analyze algorithm TEARS. Recall the assumption on n to be sufficiently large; we use it implicitly in the analysis.[2]

Overview. We begin the analysis by briefly examining the message complexity (Corollary 6.2). This rapidly yields the desired time and message complexity bounds. The main part of the analysis focuses on the correctness of the algorithm, that is, showing that each process receives a majority of the rumors. This analysis is divided into two parts.

First, we show that almost a majority of the rumors are sent to every process in the system, specifically, at least $n/2 - \frac{2n}{\log n}$ of the rumors are well distributed (Lemmas 6.4 and 6.5). Each of these rumors is sent to $\Theta(\sqrt{n})$ correct processes that then redistribute them with second level messages. This analysis requires analyzing the likelihood that the adversary's obliviously chosen schedule leads to too many messages arriving too late, that is, after a process has already sent all of its second-level messages.

Second, we show that each correct process receives sufficiently many of the *other* rumors in the system (Lemma 6.6). Here we analyze rumors that are not *well distributed*, that is, which arrive too late at too many of the processes in the system. We show that even so, sufficiently many of these rumors are forwarded to ensure majority gossip.

We then conclude the analysis in Theorem 6.7, summing up the message and time complexities.

Message Complexity. First, we estimate the number of messages sent by processes in a single step. This follows by bounding the size of the sets Π_1 and Π_2, and is crucial for estimating the overall message complexity.

LEMMA 6.1. *For every process* $\mathbf{p}$*:*

(*i*) $a - \kappa \le \Pi_1(\mathbf{p}) \le a + \kappa$
(*ii*) $a - \kappa \le \Pi_2(\mathbf{p}) \le a + \kappa$

with probability at least $1 - 1/n^3$.

PROOF. Fix a process $\mathbf{p}$ and $i \in \{1, 2\}$. Let Z be the size of $\Pi_i(\mathbf{p})$. Note that $\mathrm{E}\left[Z\right] = n(a/n) = a$. Let $\zeta = \kappa/a$. Then:

$$\Pr(Z < (1-\zeta)a) \le e^{-a\zeta^2/2} \le e^{-\kappa^2/(2a)} \le 1/n^5,$$

[2]It follows from the analysis that n needs to be large. This is because our goal is to simplify the analysis without optimizing the constants. In practice, we believe that the threshold value on n could be much smaller, however showing the result for small n would require more case-sensitive technical analysis, which we wanted to avoid for clarity of our main arguments.

as $\kappa^2/(2a) > 5\log n$. This proves that with high probability $\Pr(Z \geq a - \kappa)$. For the other direction, noting that $\zeta < 1$:

$$\Pr(Z > (1+\zeta)a) \leq e^{-a\zeta^2/3} \leq e^{-\kappa^2/(3a)} \leq 1/n^5,$$

as $\kappa^2/(3a) > 5\log n$. This proves that with high probability, $\Pr(Z \leq a + \kappa)$. Taking a union bound over the $2n$ possible values of $\mathbf{p}$ and i yields the result. $\square$

This yields, as an immediate corollary, a bound on the number of messages each process sends in each step.

COROLLARY 6.2. *Every process sends at most $a + \kappa$ point-to-point messages in each step, with probability at least* $1 - 1/n^3$.

PROOF. When process $\mathbf{p}$ decides to send messages in a step, it is either to all processes in $\Pi_1(\mathbf{p})$ or to all processes in $\Pi_2(\mathbf{p})$. The result then follows immediately from Lemma 6.1. $\square$

Using Corollary 6.2 and a direct inspection of the pseudocode, one can easily argue about the time and message complexities of algorithm TEARS; see the proof of Theorem 6.7 at the end of this section for details. The nontrivial part of the analysis is to prove the correctness of algorithm TEARS.

Correctness. We perform this analysis in four steps, captured by the four following lemmas, each proved to hold with high probability. Before formulating and proving these technical results, we introduce useful terminology.

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 is delivered in the safe period of $\mathbf{p}$. If a rumor is safe (and, by definition, received) in at least $\sqrt{n}$ nonfaulty processes, then we call such rumor *well distributed*. Note that the property of "being safe" is conditioned on receiving the rumor by the process, while the property of being "well distributed" guarantees that the rumor has been actually received by at least $\sqrt{n}$ processes in their safe periods. The intuition behind a safe rumor is that it will be sent by the process to a set of a random processes (in expectation) as a part of some second-level message, unless the process becomes faulty; therefore, a rumor that is destined to be in the safe epoch, if it is sent, in sufficiently many (random) processes will also be sent to, and received by an a/n fraction of them, on average, and thus—by being received in safe periods and so forwarded in second-level messages—it will eventually reach every nonfaulty process.

Next we describe a behavior of the oblivious adversary against our algorithm. Recall that the adversary determines a priori when processes take local steps and the latencies of possible point-to-point messages, under the restriction that the schedule satisfies d and δ. Let L be the (conceptual) set of available point-to-point links generated by each process taking its first local step, that is, $(\mathbf{q}, \mathbf{p}) \in L$ if the adversarial schedule satisfies the following: (1) $\mathbf{q}$ is allowed by the adversary to take a local step prior to failing, and (2) if $\mathbf{q}$ chooses to send a message to $\mathbf{p}$ in its first local step, then the message is delivered to $\mathbf{p}$ prior to $\mathbf{p}$ failing. Let $L_\mathbf{p}$ be the set of those links in L that have destination $\mathbf{p}$. According to the adversarial schedule, these links can be sorted according to the scheduled time of delivery of messages, should they have been sent in some processes' first local step. More precisely, $(\mathbf{q}, \mathbf{p})$ is before $(\mathbf{q}', \mathbf{p})$ in $L_\mathbf{p}$ if a message sent by $\mathbf{q}$ to $\mathbf{p}$ in its first local step would arrive before a message sent by $\mathbf{q}'$ to $\mathbf{p}$ in its first local step, if $\mathbf{q}$ and $\mathbf{q}'$ respectively should choose to send those two messages.

Note that if $\mathbf{p}$ is nonfaulty, then $n/2 < |L_{\mathbf{p}}| \leq n-1$, because a majority of processes are non-faulty.

Let $\ell_{\mathbf{p}} = \max\{\min\{|L_{\mathbf{p}}|, n/2 + \frac{n}{2a}\kappa\}, |L_{\mathbf{p}}| - \frac{2n}{a}\kappa\}$. We conceptually partition positions on each list $L_{\mathbf{p}}$, for a correct process $\mathbf{p}$, into three categories: the first $n/2$ positions are called *left-secure positions*, the ones between $n/2+1$ and $\ell_{\mathbf{p}}$ are called *right-secure*, and those bigger than $\ell_{\mathbf{p}}$ are called *insecure*. Both left- and right-secure positions together are called *secure*. We define the *weight* of process $\mathbf{p}$ (or its rumor) as the number of lists $L_{\mathbf{q}}$, for correct processes $\mathbf{q}$, for which the rumor of $\mathbf{p}$ is in a secure position.

The first of the four technical results relates the notion that $\mathbf{p}$ is in a secure position of $L_{\mathbf{q}}$ with the probability that $\mathbf{p}$ is safe in $\mathbf{q}$. Specifically, if $\mathbf{p}$ is secure (i.e., it arrives early enough in the list), then with high probability it is also safe (i.e., it is forwarded in second-level messages). This relies critically on the fact that the adversary is oblivious, that is, the adversary has to decide in advance where to schedule $\mathbf{p}$ in $L_{\mathbf{q}}$.

LEMMA 6.3. *If $\mathbf{p}$ is in a secure position of $L_{\mathbf{q}}$ (that is, $\mathbf{p}$ is in position at most $\ell_{\mathbf{q}}$ on list $L_{\mathbf{q}}$), and if $\mathbf{p}$ sends a first-level message to $\mathbf{q}$, and if $\mathbf{q}$ is not faulty, then the probability that rumor of $\mathbf{p}$ is safe in $\mathbf{q}$ is at least $1 - 3/n^4$.*

PROOF. Assume that $\mathbf{p}$ is in a secure position of $L_{\mathbf{q}}$, that is, comes no later than position $\ell_{\mathbf{q}}$, and it is received by process $\mathbf{q}$. Observe that the rumor of $\mathbf{p}$ is safe in $\mathbf{q}$ if $\mathbf{q}$ sends at least one second-level message and either of the following events occurs.

(i) There are at least κ first-level messages that arrive at process $\mathbf{q}$ after the message from $\mathbf{p}$. (This probability can be readily estimated when $\ell_{\mathbf{q}} > n/2 + \frac{n}{2a}\kappa$, that is, $\ell_{\mathbf{q}} = |L_{\mathbf{q}}| - \frac{2n}{a}\kappa$.)

(ii) There are fewer than $\mu + \kappa$ first-level messages that arrive at $\mathbf{q}$ before the one from $\mathbf{p}$. (This probability can be readily estimated when $n/2 < \ell_{\mathbf{q}} \leq n/2 + \frac{n}{2a}\kappa$.)

It follows directly from the pseudocode of the algorithm that in both cases, $\mathbf{q}$ sends a set of second-level messages after receiving a message from $\mathbf{p}$.

We first argue that $\mathbf{q}$ sends second-level messages at least once with probability at least $1 - 1/n^4$.

CLAIM 1. *Process $\mathbf{q}$ receives at least $\mu - \kappa$ first-level messages with probability at least $1 - 1/n^4$.*

PROOF OF CLAIM. Let Z be the number of first-level messages received by $\mathbf{q}$. Notice that a process $\mathbf{p}'$ sends a first-level message to process $\mathbf{q}$ with probability a/n and hence the expected number of first-level messages received by $\mathbf{q}$ is $a = 4\sqrt{n}\log n$. Therefore, since first-level messages sent to $\mathbf{q}$ are mutually independent, we conclude by a Chernoff bound: $\Pr(Z < a/2) \leq e^{-a/8} \leq n^{-4}$. Since $\mu = a/2$, the claim follows. $\square$

We next argue that if $\ell_{\mathbf{q}} > n/2 + \frac{n}{2a}\kappa$, then event (i) occurs.

CLAIM 2. *If $\ell_{\mathbf{q}} > n/2 + \frac{n}{2a}\kappa$, then there are at least κ first-level messages that arrive at process $\mathbf{q}$ after the message from $\mathbf{p}$, with probability at least $1 - 1/n^4$.*

PROOF OF CLAIM. Assume $\ell_{\mathbf{q}} > n/2 + \frac{n}{2a}\kappa$, that is, $\ell_{\mathbf{q}} = |L_{\mathbf{q}}| - \frac{2n}{a}\kappa$. We want to estimate the number of processes positioned among the last $\frac{2n}{a}\kappa$ elements of $L_{\mathbf{q}}$ that send messages to $\mathbf{q}$, as we can be sure that all such messages arrive after the message from $\mathbf{p}$ (which is secure in $\mathbf{q}$). Our goal is to show that there are at least κ such messages.

The probability that a process $\mathbf{p}'$, which is positioned among the last $\frac{2n}{a}\kappa$ elements in $L_{\mathbf{q}}$, sends a first-level message to $\mathbf{q}$ is a/n. Let Z be the expected number of such processes that send a message to $\mathbf{q}$. Then, $\mathrm{E}\left[Z\right] = 2\kappa = 64\sqrt[4]{n}\log n$.

Since the first-level messages sent to $\mathbf{q}$ by other process are mutually independent, we conclude by a Chernoff bound: $\Pr(Z < \kappa) \le e^{-\kappa/4} \le n^{-4}$. □

We now consider the second case where $n/2 + 1 \le \ell_{\mathbf{q}} \le n/2 + \frac{n}{2a}\kappa$. We show that the probability of event (ii) is also at least $1 - \frac{1}{n^4}$.

CLAIM 3. *If $n/2 + 1 \le \ell_{\mathbf{q}} \le n/2 + \frac{n}{2a}\kappa$, then there are fewer than $\mu + \kappa$ first-level messages that arrive at process $\mathbf{q}$ before the message from $\mathbf{p}$, with probability at least $1 - 1/n^4$.*

PROOF OF CLAIM. First, if the position of $\mathbf{p}$ on list $L_{\mathbf{q}}$ is smaller than $\mu + \kappa$, then the claim follows immediately (with probability 1). Assume that this is not the case.

Let Z be the number of first-level messages that arrive at process $\mathbf{q}$ prior the first level message from $\mathbf{q}$. Recall that each process sends a message to $\mathbf{q}$ with probability a/n. Since there are at least $\mu + \kappa$ processes in slots that process $\mathbf{p}$, we know that $\mathrm{E}\left[Z\right] \ge (a/n)(\mu + \kappa) \ge \frac{a^2}{2n}$ (since $\mu = a/2$).

We also know that $\mathbf{p}$ is in a secure position in $\mathbf{q}$, and hence there are at most $\ell_{\mathbf{q}}$ processes that precede $\mathbf{q}$ in the list $L_{\mathbf{q}}$. Hence, $\mathrm{E}\left[Z\right] \le \ell_{\mathbf{q}}(a/n) \le (n/2 + \frac{n}{2a}\kappa)(a/n) \le \mu + \kappa/2$.

As previously mentioned, we note that first-level messages sent to $\mathbf{q}$ are mutually independent, and so we calculate via a Chernoff bound: $\Pr(Z > (1+\alpha)\mathrm{E}\left[Z\right])$, where $\alpha = \frac{\mu+\kappa}{\mathrm{E}[Z]} - 1$. That is, $(1+\alpha)\mathrm{E}\left[Z\right] = \mu + \kappa$. There are two cases to consider. If $\alpha \le 1$, we conclude that:

$$\Pr(Z > \mu + \kappa) \le e^{-\mathrm{E}[Z]\alpha^2/3} \le e^{-\frac{(\mu+\kappa-\mathrm{E}[Z])^2}{3\mathrm{E}[Z]}} \le e^{-\frac{\kappa^2/4}{3\mathrm{E}[Z]}} \le e^{-\frac{\kappa^2/4}{3(\mu+\kappa)}} \le e^{-\frac{\kappa^2/4}{3a}} \le 1/n^4$$

because $\kappa^2/(12a) > 4\log n$. Otherwise, if $\alpha > 1$:

$$\Pr(Z > \mu + \kappa) \le \left(\frac{e^{\alpha}}{(1+\alpha)^{1+\alpha}}\right)^{\mathrm{E}[Z]} \le \left(\frac{e}{4}\right)^{\mathrm{E}[Z]} \le \left(\frac{e}{4}\right)^{\frac{a^2}{2n}} \le \frac{1}{n^4}\ ,$$

since $a^2/(2n) \ge 8\log n > 4(\log_{4/e} 2)\log n$. Both derivations are made under the assumptions that n is sufficiently large. □

Finally, by a union bound, Claims 1, 2, and 3 all hold with probability at least $1-3/n^4$, and hence the rumor of $\mathbf{p}$ is safe in $\mathbf{q}$ with probability at least $1 - 3/n^4$. □

The second key lemma demonstrates that a large fraction of the rumors have large weight. This follows from a straightforward counting argument: not too many rumors can be too late in the lists.

LEMMA 6.4. *There are more than $n/2 - \frac{2n}{\log n}$ processes of weight at least $n/\log n$.*

PROOF. We show the lemma by contradiction. Let $x \le n/2 - \frac{2n}{\log n}$ be the number of processes of weight at least $n/\log n$. Let $b > n/2$ stand for the number of correct processes. Then the total number of secure positions is *at most* $U = bx + (n-x)\cdot\frac{n}{\log n}$. (Specifically: bx is an upper bound on the number of secure positions used by processes of weight bigger than $n/\log n$, and $(n-x)n/\log n$ is an upper bound on the number of secure positions used by processes of weight at most $n/\log n$.)

On the other hand, the number of secure positions is at least $L = bn/2$, by the definition of $\ell_{\mathbf{p}}$ and the fact that $|L_{\mathbf{p}}| > n/2$. Comparing these two bounds we get a contradiction, since for $x \le n/2 - \frac{n}{\log n}$ we get

$$U = bx + (n-x)\cdot\frac{n}{\log n} < bx + 2b\cdot\frac{n}{\log n} \le bn/2 = L.$$

This completes the proof of the lemma. □

The next lemma formalizes the probabilistic intuition that any rumor of weight at least $n/\log n$ is successfully delivered to all nonfaulty processes in the second hop.

LEMMA 6.5. *All rumors of weight at least $n/\log n$ are well distributed and eventually received by each nonfaulty process, with probability at least $1 - 2/n^2$.*

PROOF. We first show that each rumor of weight at least $n/\log n$ is well distributed, with probability at least $1 - 1/n^3$. Consider a process $\mathbf{p}$ of weight at least $n/\log n$. Note that $(n/\log n) \cdot (a/n) = 4\sqrt{n}$ is the lower bound on expected number of (nonfaulty) processes $\mathbf{q}$ that have process $\mathbf{p}$ in secure position *and* are sent—and receive—the rumor from $\mathbf{p}$ in the first-level message corresponding to this position. By Chernoff bound, there are at least $2\sqrt{n}$ such processes with probability at least $1 - e^{-4\sqrt{n}/12} \geq 1 - 1/n^4$, for sufficiently large n. This immediately implies, by Lemma 6.3, that the rumor of process $\mathbf{p}$ is safe in these (nonfaulty) processes. Consequently, it is well distributed (again, with probability at least $1 - 1/n^4$). By the union bound over all rumors of weight at least $n/\log n$, they are all well distributed with probability at least $1 - n \cdot 1/n^4 = 1 - 1/n^3$.

Next, we prove that each well-distributed rumor is eventually received by each nonfaulty process, with probability at least $1 - (1/n^3 + 1/n^2)$. Recall that each rumor that is safe in process $\mathbf{p}$ is sent by process $\mathbf{p}$ to its set $\Pi_2(\mathbf{p})$. By Lemma 6.1, we know that $\Pi_2(\mathbf{p}) \geq a - \kappa$ (for all $\mathbf{p}$) with probability at least $1 - 1/n^3$. Hence, for every well distributed rumor, there are at least $\sqrt{n} \cdot (a - \kappa)$ randomly and independently selected process ids to which this rumor is sent, with probability at least $1 - 1/n^3$. There are at most n non-faulty processes, and thus the probability that there is one to which some safe rumor has not been sent is at most:

$$1/n^3 + n \cdot (1 - 1/n)^{\sqrt{n}(a-\kappa)} \leq 1/n^3 + 1/n^2 \ ,$$

as desired.

To summarize, all rumors of weight at least $n/\log n$ are well distributed, and thus sent to (and received by) every nonfaulty process, with probability at least $1 - 1/n^3 - (1/n^3 + 1/n^2) \geq 1 - 2/n^2$, for sufficiently large n. □

Note that since the number of rumors of weight at least $n/\log n$ is bigger than $n/2 - 2n/\log n$ but at most n, by Lemma 6.4, all of them are well distributed and delivered to all nonfaulty processes with probability at least $1 - \frac{2}{n^2}$, by Lemma 6.5. That is, each non-faulty process gathers more than $n/2 - 2n/\log n$ rumors with probability at least $1 - 2/n^2$. This is however not sufficient for our purpose, as we need to deliver a majority of the rumors.

It can be shown that there are a large number of rumors that are not well distributed after the first hop. Moreover, each nonfaulty 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.

Specifically, we focus on the case where there are at most $n/2$ processes of weight $n/\log n$. (Otherwise, we are already done, as per Lemma 6.5.) We show that each process receives a sufficient number of rumors from processes with weight less than $n/\log n$.

LEMMA 6.6. *Let x denote the number of process of weight at least $n/\log n$. If $x \leq n/2$, then each nonfaulty process receives at least $n/2 + 1 - x$ rumors of weight smaller than $n/\log n$ with probability at least $1 - 5/n^2$.*

PROOF. The basic idea of the proof is to count triples $(\mathbf{p}, \mathbf{q}, \mathbf{u})$ of processes such that: (i) process $\mathbf{p}$ is non-faulty and receives the rumor of process $\mathbf{u}$ in a second-level message sent by process $\mathbf{q}$, and (ii) process $\mathbf{q}$ is non-faulty and rumor of $\mathbf{u}$ is in a secure position in $L_{\mathbf{q}}$, and (iii) the rumor of $\mathbf{u}$ is of weight smaller than $n/\log n$. We call such triples *securely-relaying triples*. We estimate from above the number of securely-relaying triples for fixed processes $\mathbf{p}, \mathbf{u}$, and later the number of securely-relaying triples for fixed process $\mathbf{p}$ from below. This will allow us to estimate the number of small-weight rumors received by a non-faulty process $\mathbf{p}$, all with high probability.

Consider a rumor of process $\mathbf{u}$ of weight smaller than $n/\log n$ and a nonfaulty process $\mathbf{p}$. Rumor $\mathbf{u}$ is received by at most $6 \cdot \frac{n}{\log n} \cdot \frac{a}{n} = 6 \cdot \frac{a}{\log n}$ nonfaulty processes $\mathbf{q}$ having $\mathbf{u}$ in a secure position, with probability at least $1 - 2^{-5a/\log n} \geq 1 - 1/n^3$, by a Chernoff bound. Therefore, a nonfaulty process $\mathbf{p}$ receives the rumor of $\mathbf{u}$ from at most $6 \cdot \frac{6a}{\log n} \cdot \frac{a}{n} \leq \frac{36a^2}{n \log n}$ nonfaulty processes $\mathbf{q}$ having $\mathbf{u}$ in a secure position, with probability at least $1 - 1/n^3 - 2^{-30a^2/(n\log n)} \geq 1 - 2/n^3$, again by Chernoff bound and a union bound. Hence, for fixed processes $\mathbf{p}, \mathbf{u}$, the number of securely relaying triples $(\mathbf{p}, \mathbf{q}, \mathbf{u})$ is at most $\frac{36a^2}{n \log n}$ with probability at least $1 - 2/n^3$.

Next we estimate the number of securely-relaying triples with only the first coordinate fixed on $\mathbf{p}$, for a nonfaulty process $\mathbf{p}$. Notice that every correct process sends second-level messages at least once with probability at least $1 - 1/n^3$, by Claim 1 in the proof of Lemma 6.3. Under this condition, a nonfaulty process $\mathbf{p}$ should expect to receive second-level messages from at least $\frac{n}{2} \cdot \frac{a}{n}$ processes; thus, by a Chernoff bound, process $\mathbf{p}$ receives second-level messages from at least $\frac{1}{2} \cdot \frac{n}{2} \cdot \frac{a}{n} = \mu/2$ relaying non-faulty processes $\mathbf{q}$, with probability at least $1 - e^{-\mu/8} \geq 1 - 1/n^3$. After removing the conditioning event, the probability that process $\mathbf{p}$ receives second-level messages from at least $\mu/2$ relaying non-faulty processes $\mathbf{q}$ is at least $1 - 1/n^3 - 1/n^3 = 1 - 2/n^3$.

Notice that there are at least $n - x$ processes with weight less than $n/\log n$, and for a given relay, there are at most $n/2 - 1$ insecure positions. Hence, for a given relay, there are at least $n/2 + 1 - x$ processes of weight less than $n/\log n$ in secure positions. Each of these sends a first-level message to that relay with probability a/n.

Thus, since there are at least $\mu/2$ relayed messages received by $\mathbf{p}$ (with probability at least $1 - 1/n^3$), we know that the (at least) $\mu/2$ second-level messages received by $\mathbf{p}$ collectively carry at least $\frac{1}{2} \cdot \frac{\mu}{2} \cdot (n/2 + 1 - x)\frac{a}{n} = \frac{a^2}{8n} \cdot (n/2 + 1 - x)$ "copies" of rumors with weight smaller than $n/\log n$ stored on secure positions in the relaying processes, with probability at least $1 - e^{-(\mu/2)\cdot(n/2+1-x)\cdot(a/n)/8} \geq 1 - 1/n^3$, by a Chernoff bound (for sufficiently large n). Consequently, the probability that each nonfaulty process $\mathbf{p}$ is counted in at least $\frac{a^2}{8n} \cdot (n/2 + 1 - x)$ different securely relaying triples $(\mathbf{p}, \mathbf{q}, \mathbf{u})$, is at least $1 - 2/n^3 - 1/n^3 = 1 - 3/n^3$.

Using these upper and lower bounds, we conclude that a nonfaulty process $\mathbf{p}$ receives at least

$$\frac{(a^2/8n) \cdot (n/2 + 1 - x)}{(36a^2/n \log n)} \geq \frac{n}{2} + 1 - x$$

different rumors $\mathbf{u}$ of weight smaller than $n/\log n$ (for sufficiently large n), with probability at least $1 - 2/n^3 - 3/n^3 \geq 1 - 5/n^3$. By union bound, this holds for every nonfaulty process $\mathbf{p}$ with probability at least $1 - 5/n^2$. $\square$

We now conclude the following.

THEOREM 6.7. *Algorithm* TEARS *completes majority gossip with time complexity $O(d + \delta)$ and message complexity $O(n^{7/4} \log^2 n)$, with high probability subject to an oblivious adversary.*

PROOF. The correctness is guaranteed by Lemmas 6.4, 6.5, and 6.6 with probability at least $1 - \frac{2}{n^2} - \frac{5}{n^2} \geq 1 - \frac{7}{n^2}$. It remains to prove the complexity bounds. Let n be sufficiently large.

Time Complexity. All first-level messages arrive by time $d + \delta$, by time $d + 2\delta$ every second-level message is sent, and it arrives by time $2d + 2\delta$. Hence, the bound $O(d + \delta)$ follows.

Message Complexity. Each nonfaulty process sends $a + \kappa \leq 10\sqrt{n}\log n$ first-level messages, by Corollary 6.2, and thus receives at most $20\sqrt{n}\log n$ first-level point-to-point messages on average (because of the upper bound $n/2$ on the number of failures), which by Chernoff bound guarantees receiving no more than $40\sqrt{n}\log n$ received messages, all with high probability. Therefore it sends fewer 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 Corollary 6.2 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. □

7. 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 the following.

(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 correct 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 Canetti-Rabin framework to produce an efficient consensus protocol.

We begin by recalling the Canetti-Rabin framework introduced in Canetti and Rabin [1993], and we follow the simplified presentation of Attiya and Welch [2004, 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 Attiya and Welch [2004] 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 Attiya and Welch [2004], get-core is implemented by three sequential phases of all-to-all communication in which

```
consensus($value$)
▷ Variables:
$rnd \in \mathbb{N}$                 ▷ round counter, initially $0$
$vote \in \{0,1\}$
$prefer \in \{0,1,\perp\}$
$log$                     ▷ array of sets of ⟨ value, id ⟩ pairs, initially empty

$vote \leftarrow value$

repeat :
    $rnd \leftarrow rnd + 1$
    $log[rnd] \leftarrow$ get_core($vote, log, rnd$)
    if $\exists v$ such that $|\{j : \langle v, j\rangle \in log[rnd]\}| \geq \lfloor n/2 \rfloor + 1$
        then   $prefer \leftarrow v$
        else   $prefer \leftarrow \perp$
    if $\exists v$ such that $\forall v' \neq v,\ \langle v', *\rangle \notin log[rnd]$
        then   decide($v$)
    $rnd \leftarrow rnd + 1$
    $log[rnd] \leftarrow$ get_core($prefer, i, rnd$)
    $rnd \leftarrow rnd + 1$
    $log[rnd] \leftarrow$ get_core(random($1/n$), $log, rnd$)
    if $\exists v$ such that $\forall v' \neq v,\ \langle v', *\rangle \notin log[rnd-1]$
        then   $vote \leftarrow v$
    elseif $\exists j$ such that $\langle 0, j\rangle \in log[rnd]$
        then   $vote \leftarrow 0$
        else   $vote \leftarrow 1$
```

Fig. 5. Asynchronous consensus protocol that tolerates an oblivious adversary.

each process sends all the votes it has received in that round of voting to everyone else, leading to $O(n^2)$ message complexity.

Efficient (majority) gossip can be used to reduce the message complexity. The detailed protocol is included in Figures 5 and 6.

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 nonfailed process begins to gossip; this follows immediately from the epidemic spread of the initiator's rumor. Similarly, with algorithm SEARS (respectively, TEARS), every nonfailed process begins to gossip within $O(\frac{1}{\varepsilon}(d+\delta))$ time (respectively, $O(d+\delta)$). 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 this explanation, we conclude the following theorem.

```
get_core($v, log, rnd$)
▷ Variables:
$R$                                              ▷ a set of ⟨ value, id ⟩ pairs
$\ell \in \{1, 2, 3\}$
$iteration, iter$                                ▷ array of sets of ⟨ value, id ⟩ pairs
$rumors_{r,\ell}$, where $r \in \mathbb{N}$, $\ell \in \{1, 2, 3\}$       ▷ a set of $\langle R,\ log,\ iteration\rangle$ triples

$R \leftarrow \{\langle v, i\rangle\}$

for $\ell = 1$ to $3$ do
    Start asynchronous $rumors_{rnd,\ell} \leftarrow$ gossip($\langle R, log, iteration\rangle)_{rnd,\ell}$
    Wait until either:
        (1) $|rumors_{rnd,\ell}| \geq \lfloor n/2\rfloor + 1$
            $iteration[\ell] \leftarrow \bigcup\{R' : \langle R', *, *\rangle \in rumors_{rnd,\ell}\}$
            $R \leftarrow R \cup iteration[\ell]$

        (2) $\exists r > rnd, j \in \{1, 2, 3\}$ such that $|rumors_{r,j}| \geq 1$
            Choose $log$ such that $\langle *, log, *\rangle \in rumors_{r,j}$
            return $log[rnd]$

        (3) $\exists j \in \{1, 2, 3\},\ j > \ell$ such that $|rumors_{rnd,j}| \geq 1$
            Choose $iter$ such that $\langle *, *, iter\rangle \in rumors_{rnd,j}$
            $iteration[\ell] \leftarrow iter[\ell]$
            $R \leftarrow R \cup iteration[\ell]$

return $R$
```

Fig. 6. get_core routine for process i. All sets and arrays begin initially empty.

THEOREM 7.1. *For an oblivious adversary and a minority of failures, in expectation,*

—*Algorithm CR*-EARS *has $O(\log^2 n(d+\delta))$ 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.*

8. CONCLUSIONS

This article studies the complexity of gossip in an asynchronous, message-passing distributed system subject to processes crash failures. We have demonstrated that gossip is inherently inefficient in the context of an adaptive adversary, but that it is possible to develop efficient, randomized, asynchronous gossip algorithms subject to an oblivious adversary. The main challenge of developing such algorithms is overcoming the unknown bound on communication delay and process speed, both of which are typically used in synchronous algorithms to decide when a process should stop sending messages or whether a process has crashed. Under an oblivious adversary, our gossip algorithms can be used to implement efficient asynchronous randomized consensus protocols; one variant terminates in constant time and has strictly subquadratic message complexity. This last result was achieved by considering a weaker version of gossip, called majority gossip.

One interesting open question is whether our gossip algorithms are optimal. What are the lower bounds on time and message complexities for gossiping under an oblivious adversary? In fact, notice that both of our gossip protocols (but not the weaker majority gossip protocol) have message complexity that depends on d and δ, that is, the latency of the network. Can this be avoided? Practical systems often attempt to adjust the frequency of sending gossip in order to approximate a synchronous environment. When network latencies are predictable or readily measurable, such strategies often yield message complexity that is independent of the network latency. Are there gossip protocols that can achieve this adaptivity, in the worst-case, or is a dependence on network latency inherent?

There are many interesting open questions related to majority gossip. This weaker gossip primitive appears easier to implement efficiently, and yet sufficiently powerful to solve a variety of problems. This conjecture, which we first proposed in Georgiou et al. [2008], has been further supported by Censor Hillel and Shachnai [2010] in their work on partial information spreading. Over the long-term, we would like to understand where we can use majority gossip, instead of full gossip, and we expect to find a variety of new applications for majority gossip, for example, load balancing and distributed shared memory implementations.

There are three immediate questions. First, can the message complexity of majority gossip be improved further? While the protocol presented here ensures sub-quadratic message complexity, there seems no inherent reason that this cannot be further reduced. Moreover, any improvements in the message complexity of majority gossip will immediately translate to more efficient constant-round consensus protocols. The question of optimal message complexity for asynchronous, oblivious consensus protocols remains open. The second immediate question regrading majority gossip is whether it is possible to show a separation between majority gossip and full gossip. Can we show that majority gossip is, in fact, inherently more efficient that full gossip? How efficient is majority gossip in the context of an adaptive adversary? Finally, the third question regarding majority gossip is whether there are any efficient *deterministic* majority gossip algorithms. In synchronous systems, many randomized gossip protocols can be derandomized via expander graphs and other similar techniques. Majority gossip seems potentially amenable to such techniques in an asynchronous environment.

Finally, an important open question is the communication complexity of asynchronous gossip, that is, the total number of bits exchanged in a given computation. In this article, we have focused on message complexity, as in many applications aggregation and compression techniques allow for fixed-sized messages. In some applications, however, this is not possible, and the overall communication complexity becomes critical. In real systems, bandwidth is often the limiting resource and minimizing the communication complexity is the best way to ensure efficient operation.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their comments that have helped in significantly improving the manuscript.

REFERENCES

Aspnes, J. 2003. Randomized protocols for asynchronous consensus. *Distrib. Comput. 16*, 2–3, 165–175.

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

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

Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y. 1999. Bimodal multicast, *ACM Trans. Comput. Syst. 17*, 2, 41–86.

Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. 2006. Randomized gossip algorithms. *IEEE Trans. Inf. Theory, 52*, 6, 2508–2530.

Canetti, R. and Rabin, T. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*. 42–51.

Censor Hillel, K. and Shachnai, H. 2010 Partial information spreading with application to distributed maximum coverage. In *Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC)*. 161–170.

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

Chlebus, B. S. and Kowalski, D. R. 2006a. Robust gossiping with an application to consensus. *J. Comput. Syst. Sciences, 72*, 1262–1281.

Chlebus, B. S. and Kowalski, D. R. 2006b. Time and communication efficient consensus for crash failures. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*. 314–328.

Chor, B. and Dwork, C. 1989. Randomization in Byzantine agreement. In *Advances in Computing Research 5: Randomness and Computation*. 443–497.

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

Dwork, C., Lynch, N., and Stockmeyer, L. 1988. Consensus in the presence of partial synchro. *ACM, 35*, 2, 288–323.

Eugster, P., Guerraoui, R., Handurukande, S., Kermarrec, A.-M., and Kouznetsov, P. 2003. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst. 21*, 4.

Georgiou, Ch., Gilbert, S., Guerraoui, R., and Kowalski, D. R. 2008. On the complexity of asynchronous gossip. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*. 135–144.

Georgiou, Ch., Kowalski, D. R., and Shvartsman, A. A. 2005. Efficient gossip and robust distributed computation. *Theoret. Comput. Sci., 347*, 130–166.

Georgiou, Ch. and Shvartsman, A. A. 2008. *Do-All Computing in Distributed Systems: Cooperation in the Presence of Adversity*, Springer.

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

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

Karp, R., Schindelhauer, C., Shenker, S., and Vocking, B. 2000. Randomized rumor spreading. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*. 565–574.

Kempe, D., Kleinberg, J., and Demers, A. 2004. Spatial gossip and resource location protocols. *ACM, 51*, 943–967.

Kermarrec, A. M., Massoulie, L., and Ganesh, A. 2003. Probabilistic reliable multicast in ad hoc networks. *IEEE Trans Paral Distr. Syst.* 14.

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

Mitzenmacher, M. and Upfal, E. 2005. *Probability and Computing*. Cambridge University Press.

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

Van Renesse, R., Minsky, Y., and Hayden, M. 1998. A gossip-style failure detection service. In *Proceedings of the IFIP Int-l Conference on Distributed Systems Platforms and Open Distributed Processing*. 55–70.

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

Received May 2011; revised June 2012 and January 2013; accepted January 2013