Meeting the Deadline: On the Complexity of Fault-Tolerant Continuous Gossip *

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

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

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

ABSTRACT

In this paper, we introduce the problem of *Continuous Gossip* in which rumors are continually and dynamically injected throughout the network. Each rumor has a deadline, and the goal of a continuous gossip protocol is to ensure good "Quality of Delivery," i.e., to deliver every rumor to every process before the deadline expires. Thus, a trivial solution to the problem of Continuous Gossip is simply for every process to broadcast every rumor as soon as it is injected. Unfortunately, this solution has a high per-round message complexity. Complicating matters, we focus our attention on a highly dynamic network in which processes may continually crash and recover. In order to achieve good per-round message complexity in a dynamic network, processes need to continually form and re-form coalitions that cooperate to spread their rumors throughout the network. The key challenge for a Continuous Gossip protocol is the ongoing adaptation to the ever-changing set of active rumors and non-crashed process. In this work we show how to address this challenge; we develop randomized and deterministic protocols for Continuous Gossip and prove lower bounds on the per-round message-complexity, indicating that our protocols are close to optimal.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: [fault tolerance]

General Terms

Algorithms, Theory

*A full version of the paper is available at the first author's website.

†The work of this author was supported in part by research funds from the University of Cyprus.

‡The work of this author was supported by the Engineering and Physical Sciences Research Council [grant number EP/G023018/1].

PODC'10, July 25–28, 2010, Zurich, Switzerland.

Keywords

Gossip, crashes and restarts, dynamic rumor injection, expander graphs

1. INTRODUCTION

Disseminating information lies at the core of distributed computing. *Gossiping* is a fundamental mechanism for achieving efficient data dissemination [20]. The *Gossip problem* is typically defined as follows: a set of n processes each receive an initial piece of information, called a *rumor*; each process wants to learn all the other rumors. Gossip protocols have been used as building blocks in numerous areas of distributed computing, including distributed consensus [5, 12, 13], database consistency [8], cooperative task computing [14], failure detection [23], group communication [1, 11, 18], and resource location [17].

Continuous Gossip. In this paper, we focus on a generalized gossip problem, *Continuous Gossip*, that we believe better captures the requirements of data dissemination in large-scale, dynamic distributed systems. Continuous gossip differs from traditional gossip in the following ways:

Continuous: Most gossip algorithms tend to focus on *single-shot* versions of the data dissemination problem. In this paper, we study *Continuous Gossip* where rumors are injected dynamically, *at any process*, *at any time*. By contrast to traditional gossip, an execution is of unbounded duration, and there is no bound on the number of rumors that may be injected during the execution.

Real-time: We focus on a *real-time* variant of the gossip problem: every rumor has a strict deadline by which it must be delivered. Some rumors may have short deadlines, and need to be delivered in $O(1)$ time; other rumors may have longer deadlines allowing for slower delivery.

Targeted: Most gossip protocols distribute every rumor to every process. In this paper, each rumor has a specified set of destinations that may be much smaller. For example, in a video streaming system, the source may wish to send a stream (i.e., a set of rumors) to a certain set of viewers.

Continuous gossip can be used as a basic building block for designing distributed services. For example, consider implementing a *publish-subscribe* service in which there are a set of topics, each associated with a group of processes that subscribe to that topic: whenever a message is published, the publisher injects a rumor with the specified group as the set of destinations; different topics may have different deadlines—for example, a topic representing a video stream

may have short deadlines, while a topic representing an e-mail listerv may have longer deadlines.

Fault Tolerance. The efficiency of gossip is greatly affected by faults. The fault-tolerance of gossip protocols has been widely studied in the context of process crashes, link failures, transient disconnections, message omission faults, Byzantine failures, etc. (See, for example, [15, 19, 20].) Here, we focus on a dynamic network in which processes may *crash* and *restart* at any time. Such networks are particularly challenging since there is no guarantee that any given subset of processes will remain active. In fact, we assume no lower bound on the number of active processes; at any given time, all the active processes may fail!

Metrics. There is a trivial solution to the problem of continuous gossip: each rumor is sent immediately to the desired set of destinations. When there are not many rumors in the system, or when each rumor is destined for only a small number of destinations, this simple "protocol" may be quite efficient, requiring only one message per rumor-destination. And for rumors with a very short deadline (e.g., one round), there may be no other feasible solution.

However, when there are a large number of rumors and a large number of destinations, such a simple approach may require quadratic (w.r.t the number of processes) $\Theta(n^2)$ messages per round. Thus, our goal is to develop a continuous gossip protocol that minimizes the per-round message complexity, i.e., the maximum number of messages sent in any round. (Notice that since we are considering an unbounded execution and an unbounded number of rumors, it does not make sense to consider *total* message complexity, as that may also be unbounded.) As hinted above, the per-round message complexity will depend significantly on the length of the deadline: very short deadlines will lead to (unavoidably) high per-round message complexity, while longer deadlines allow for more efficient data dissemination.

Contributions. To the best of our knowledge, this paper is the first to consider the complexity of gossip subject to dynamic crashes and restarts and dynamic injection of deadline-based rumors. To summarize our contributions:

1. Problem Definition: We formulate the *Continuous Gossip* problem, and define an acceptable *Quality of Delivery* (QoD) that defines the correctness of Continuous Gossip.

2. Lower Bounds: We prove lower bounds on the per-round message complexity of continuous gossip: if rumors have a deadline of d, then every *deterministic* protocol has a per-round message complexity of $\Omega(n^{1+1/d}/d)$; every *randomized* protocol has, with high probability, a per-round message complexity of $\Omega(n^{1+1/d}/d^2)$. These lower bounds imply that the message complexity is strongly related to the length of the deadline of the rumors: for very short deadlines, linear per-round message complexity is impossible; yet for slightly longer deadlines, the lower bound is compatible with achieving subquadratic per-round message complexity. The randomized and deterministic algorithms we develop are close to optimal with respect to these lower bounds. In fact, for $d = \log^{O(1)}(n)$, they are optimal within log factors.

3. Randomized Algorithm: We develop an efficient randomized algorithm for continuous gossip that sends at most $O(n^{1+5\sqrt{2/d_m}} \log^{O(1)}(n))$ messages per-round, where d_m is the minimum deadline of any rumor active in the system. The key to achieving good efficiency is for processes with *active* rumors (i.e., those whose deadline has not expired) to collaborate, sharing the work of distributing the rumors. Thus, the algorithm relies on two random mechanisms that execute concurrently: the first mechanism discovers other collaborators, i.e., processes with active rumors; the second mechanism distributes the rumors to their destinations. At any given time, any (or all) of the collaborating processes may crash, and the remaining processes must finish the job.

Notice that a key challenge here is coping with crashes and restarts. In the absence of failures, continuous gossip is easy: one process is designated as the coordinator, which collects and distributes the rumors so as to meet all the deadlines. Even simple crash failures are easier to tolerate. For example, gossip algorithms typically attempt to build a low-degree overlay on top of the fully connected network, enabling efficient dissemination of information. As the number of failures increases, in order to prevent the overlay from becoming disconnected, protocols typically increase the degree of the overlay, i.e., increase the amount of communication among the surviving processes. As long as the increase in messages is slower than the rate of failures, good message complexity can be maintained. This strategy fails entirely when processes can restart: if processes send more messages to compensate for failures, the failed nodes may also restart, overwhelming the network with messages. Since an execution of continuous gossip may be arbitrarily long, it does not make sense to bound the number of crash failures; hence we must cope with the case where processes can restart.

4. Adaptivity: We show that the randomized algorithm is *adaptive*. When there are a large number of rumors or a large number of destinations, the per-round message complexity is always $O(n^{1+5\sqrt{2/d_m}} \log^{O(1)}(n))$; yet when there are a small number of rumors or a small number of destinations-per-rumor, the trivial "direct transmission" protocol where each source sends its own rumors directly to the destinations may be better. Our randomized protocol adapts to these "sparse" situations, achieving a per-round message complexity that is almost as good as the "direct transmission" solution. More specifically: if $\{\rho_1, \rho_2, \ldots, \rho_k\}$ are the set of active rumors in some round, and if $\rho_j.D$ is the set of destinations for rumor ρ_j, then our algorithm never sends more than $O\left(\sum_{j=1}^{k} |\rho_j.D| \log^{O(1)}(n)\right)$ messages.

This property of *adaptivity* is hard to achieve precisely because of dynamic rumor injection: if rumors are only injected at a single process, then the optimal behavior of that process is to simply send the rumor directly to everyone. By contrast, if rumors are injected widely throughout the network, then processes must perform a cooperative protocol to reduce the message complexity. Unfortunately, to distinguish these cases, the processes need to exchange information amongst themselves, leading to increased message complexity. In order to adapt to the message injection pattern, the algorithm we present is efficient in its search for collaborating processes, and in the sharing of work among collaborators, introducing relatively little message overhead.

5. Deterministic Algorithm: We show how to *de-randomize* the algorithm above, replacing the random choices with expander graphs. This leads to a deterministic algorithm that sends at most $O(n^{1+6/\sqrt[3]{d_m}} \log^{O(1)}(n))$ messages per round, where d_m is the minimum deadline of any rumor active in the system. (While the deterministic version can also be made

adaptive, we omit that aspect.) We show a new property of the specified expander graphs relevant to the crash-restart environment. The resulting protocol is quite involved, as in a deterministic environment it is significantly harder to coordinate collaborating processes: due to failures, they may not be as well connected; it is also harder to evenly distribute the work of disseminating rumors among the collaborators.

Other related work. There is a large amount of literature dedicated to fault-tolerant gossip in different settings; we limit our discussion only to the most relevant prior work.

The gossip problem has frequently been considered in relation to random, epidemic communication (see for example [10, 16, 17, 18]). In this context, the problem is also know as *rumor spreading* and the protocols usually use a simple epidemic approach: each process periodically sends its rumor—along with any new rumors it has learned—to another randomly selected process. Karp et al [16] showed, using a synchronous round-based epidemic protocol (a process sends a rumor to one other process in each round), that a single rumor can be disseminated in $O(\log n)$ rounds using $O(n \log \log n)$ messages with high probability, n being the number of processes.

The best (to-date) deterministic synchronous crash-tolerant algorithm (for a complete communication network) for one-shot n-rumor gossip is due to Chlebus and Kowalski [6]. Their algorithm achieves $O(\log^3 n)$ time complexity and $O(n \log^4 n)$ total message complexity, even if up to $n-1$ processes may crash. In [19], Kowalski and Strojnowski studied the impact of faults on the total message complexity of n-rumor deterministic gossip problem under several failure classes: crashes, message omissions, authenticated Byzantine and Byzantine faults. Focusing on solutions with constant time complexity, they showed that crashes cost more messages than non-faulty processes, however polynomially fewer messages than more severe types of failures like omission and (authenticated) Byzantine. According to our knowledge, restarts were not considered in the context of the complexity of one-shot distributed gossip. The survey by Pelc [20] together with the book by Hromkovic et al. [15] provide a presentation of gossiping in fault-prone distributed networks (under various network topologies).

As in the present work, for the purposes of their algorithms, Chlebus and Kowalski [6] and Kowalski and Strojnowski [19] defined and used graphs with specific fault-tolerant properties. Different kinds of graphs with expansion properties were studied before in the context of fault-tolerant communication in message-passing systems and networks [4, 5, 6, 9, 14] and shared memory [7]. In this work we study a new fault-tolerant property of expander graphs, in the context of a tradeoff between the initial number of non-faulty nodes and the diameter of the connected component induced by some large subset of non-faulty nodes.

2. MODEL

Distributed Setting. We consider a synchronous, message-passing system of n crash-prone processes (where n is fixed and known a priori). Processes have unique ids from the set $[n] = \{1, 2, \ldots, n\}$. Each process can communicate directly with every other process (i.e., the underlying communication network is a complete graph); messages are not lost or corrupted.

Rounds. The computation proceeds in synchronous rounds. In each round, each process can: (i) send point-to-point messages to selected processes, (ii) receive a set of point-to-point messages sent in the current round, and (iii) perform some local computation (if necessary). We assume that there is no global clock available to the processes, i.e., rounds are not globally numbered. Any reference to a global round number is only for the purpose of presentation or algorithm analysis.

Crashes/Restarts. Processes may crash and restart dynamically as an execution proceeds. Each process is in one of two states: either `alive` or `crashed`. When a process is crashed, it does not perform any computation, nor does it send or receive any messages. Processes have no durable storage, and thus when a process restarts, it is reset to a default initial state consisting only of the algorithm and $[n]$. Each process can only crash or restart once per round.

When a process p crashes in round t, some of the messages sent by p in round t may be delivered, and some may be lost. Similarly, when a process p restarts in round t, some of the messages sent to p may be delivered and some may be lost. (Recall that no message is sent/received by a process that is crashed during the whole round.)

We denote by $crash(p, t, X)$ the crash event for process p in round t, where X is the (possibly empty) set of processes that are allowed to receive messages from p in round t. We denote by $restart(p, t, Y)$ the restart event for process p in round t, where Y is the (possibly empty) set of processes that are allowed to deliver messages to p in round t.

We say that a process p is **continuously alive** in the period $[t_a,\ t_b]$ if: (a) process p is `alive` at the beginning of round t_a and at the end of round t_b, and (b) for every $t \in [t_a,\ t_b]$, there are no $crash(p, t, \cdot)$ events.

Rumors. Rumors are dynamically injected into the system as the execution proceeds. A rumor ρ consists of a 4-tuple $\langle z,\ D,\ d,\ p \rangle$, where z is the data to be disseminated, $D \subseteq [n]$ is the set of processes to which z must be sent (destination set), d is the deadline duration by which the rumor must be delivered, and p is the process at which the rumor is injected; we call p the *source* of rumor ρ. We denote by $inject(\rho, t)$ the event where rumor ρ is injected in round t.

We assume that at most one rumor per round is injected at each process. Rumors are injected at the beginning of a round, and only at processes that are alive throughout the round. The basic expectation is that a rumor $\langle z,\ D,\ d,\ p \rangle$ injected in round t should be delivered by the end of round $t + d$ to all processes in D that are alive. Due to continuous crashes and restarts, this goal (delivery to all processes in D that are alive) must be specified in more detail; see Section 3.

Adversary. We model crash/restarts and rumor injection via a *Crash-and-Restart-Rumor-Injection (CRRI) adversary*. Adversary $CRRI$ consists a set of *adversarial patterns* $\mathcal{A} = \{\mathcal{F},\ \mathcal{R}\}$, where $\mathcal{F}$ denotes a set of crash/restart events and $\mathcal{R}$ denotes a set of rumor injection events.

We assume that every adversarial pattern $\mathcal{A} \in CRRI$ satisfies natural well-formedness conditions, e.g., a $crash(p, t, X)$ event cannot occur if p is already `crashed` in round t, and a $restart(p, t, Y)$ event cannot occur if p is already `alive`. We also assume that the adversary "knows" the algorithm being executed by the processes, meaning that the set of adversarial patterns may depend on the algorithm. However, in the case of randomized algorithms, the adversary is not aware of the random choices made

during the execution. That is, the adversary is *oblivious*: it chooses the adversarial pattern prior to the execution (and cannot change the pattern once the execution begins).

Complexity Metrics. Typically, message complexity accounts for the *total* number of point-to-point messages sent during a given computation. (A multicast to k processes counts as k point-to-point messages.) However, in this work we allow for computations to have unbounded duration, and rumors may be injected into the system over an unbounded time period; thus counting the total number of messages sent in the entire computation is not meaningful. Instead, we focus on the number of messages sent *per round*.

More formally, let Det be a *deterministic* algorithm operating under adversary $CRRI$. For an adversarial pattern $\mathcal{A} \in CRRI$, define $M_t(Det, \mathcal{A})$ to be the number of messages sent by Det in round t. We say that algorithm Det has **per-round message complexity** at most $M(Det)$ if $\forall t, \forall \mathcal{A} \in CRRI,\ M_t(Det, \mathcal{A}) \leq M(Det)$.

Similarly, the per-round message complexity for a *randomized* algorithm is a bound on the number of messages that, with high probability, are sent in a round. More formally, we say that a *randomized* algorithm $Rand$ operating under adversary $CRRI$ has per-round message complexity at most $M(Rand)$, if for every round t, for every $\mathcal{A} \in CRRI$, with high probability, $M_t(Rand, \mathcal{A})$ is at most $M(Rand)$.

3. QUALITY OF DELIVERY (QOD) AND CONTINUOUS GOSSIP

We now describe the desired *quality-of-delivery* (QoD), that is, the requirements on when a rumor will be delivered. Ideally, we would like every rumor ρ injected in the system to be learned by all processes in $\rho.D$. Moreover, each rumor should be delivered before it expires. However, this is not always possible: for example, a process $q \in \rho.D$ may be `crashed` throughout the duration of a rumor's lifetime.

A natural (weaker) quality-of-delivery guarantee is as follows: if $\rho = \langle z,\ D,\ d,\ p\rangle$ is a rumor injected in round t, and if p is continuously alive in the period $[t+1,\ t+d]$, then ρ is delivered to every process $q \in D$ that is continuously alive in *at least one round* in the period $[t+1,\ t+d]$. While it is possible to achieve such a *strong quality-of-delivery* guarantee, unfortunately this unavoidably requires high message complexity in our context. This is due to the following fact:

FACT 1. *Any deterministic or randomized algorithm that guarantees strong QoD under adversary CRRI must disseminate each injected rumor immediately.*

Otherwise, if process p does not immediately send the rumor injected in round t to every other process in the rumor's destination set in round $t+1$ (i.e., as soon as possible), then, for example, a process q that was alive in round $t+1$, but not in the period $[t+2,\ t+d]$, will not receive the rumor, violating the strong QoD requirement. Thus, any such protocol will have high per-round message complexity.

Instead, we say that a rumor $\rho = \langle z,\ D,\ d,\ p\rangle$ injected at p in round t is **admissible** for $q \in D$ if both p and q are continuously alive in the period $[t+1,\ t+d]$. We say that a protocol ensures **satisfactory quality-of-delivery** or simply **quality of delivery (QoD)** if it delivers every rumor $\rho = \langle z,\ D,\ d,\ p\rangle$ injected at p in round t that is admissible for $q \in D$ to process q by the end of round $t+d$.

Basically, for a rumor to be admissible, the adversary needs to leave a sufficiently large "window of opportunity" for p to deliver the rumor to q. This yields more efficient (and more practical) protocols. Note that for randomized algorithms, it is natural to expect the QoD guarantee to hold with probability 1. In other words, we expect both deterministic and randomized algorithms to *guarantee* QoD.

We say that an algorithm Alg (randomized or deterministic) solves the Continuous Gossip problem if it guarantees QoD under any adversarial pattern $\mathcal{A} \in CRRI$. Our goal is to design randomized and deterministic algorithms for the Continuous Gossip problem with subquadratic message complexity (that is, $M(Alg) = o(n^2)$).

4. LOWER BOUNDS

We begin our study of efficient solutions (with respect to message complexity) for the Continuous Gossip problem by looking at lower bounds. We focus on adversarial patterns in which the adversary $CRRI$ injects rumors that must be delivered to all processes (that is, the destination set is $[n]$) and with uniform deadlines (that is, all rumors have the same deadline). We first show a lower bound for deterministic and then a lower bound for randomized algorithms.

THEOREM 1. *Every deterministic algorithm Det guaranteeing QoD for rumors with destination set $[n]$ and deadline d with at most $M(Det)$ per-round message complexity must have $M(Det) = \Omega(n^{1+1/d}/d)$.*

THEOREM 2. *Every randomized algorithm Rand guaranteeing QoD for rumors with destination set $[n]$ and deadline d with at most $M(Rand)$ per-round message complexity with probability bigger than $1-\min\{1/(2d), 2/n\}$, must have $M(Rand) = \Omega(n^{1+1/d}/d^2)$.*

Observe that the above lower bound results suggest that for small deadlines, a superlinear per-round message complexity is inevitable. (This is in contrast to the model with no failures.) Furthermore, this has motivated our search for randomized and deterministic continuous gossip solutions with subquadratic per-round message complexity. The randomized and deterministic algorithms we develop in the following two sections achieve such subquadratic complexity.

5. RANDOMIZED SOLUTION

We now describe and analyze a randomized algorithm, called rand-gossip, for the Continuous Gossip problem.

5.1 Basic Strategy

Rumors may be injected in any round, at any process, with different deadlines and different sets of destinations. We partition these rumors, as they are injected, into sets of rumors that were: (i) injected in the same round; (ii) have approximately equal deadlines; and (iii) have approximately the same number of destinations. For each such set of rumors, we spawn a separate instance of routine fixed-rand-gossip. We ensure that there are at most $O(\log^4 n)$ instances of fixed-rand-gossip running in each round, and hence this partitioning increases the per-round message complexity by at most a polylogarithmic factor. (Some log-factor improvements can be gained by avoiding this partitioning, but the resulting protocol is significantly more complicated.)

The pseudocode for the main rand-gossip routine is described in Algorithm 1. The rumor's deadline is rounded down to the nearest power of two, and truncated at $\Theta(\log^2 n)$. (Every rumor is delivered within $\Theta(\log^2 n)$

rounds, regardless of the deadline – there is no benefit to deadlines longer than $\Theta(\log^2 n)$.) The size of the rumor's destination set is rounded up to the nearest power of two. The process then spawns a new instance of fixed-rand-gossip. Thus, there may be many instances of fixed-rand-gossip running in parallel. Messages from each instance are distinguished by adding certain control bits (such as *dline* and *dsize*). For clarity, we treat each instance as exchanging messages on its own private network. Therefore, for the remainder of this section, we present and analyze a single instance of the fixed-rand-gossip routine, concluding with an analysis of the per-round message complexity of the entire randomized algorithm in Theorem 3.

```
1  procedure rand-gossip(rumor ρ)_i
2      dline ← min{25 log^2 n, ρ.d}
3      dline ← 2^⌊log dline⌋
4      dsize ← 2^⌈log |ρ.D|⌉
5      spawn fixed-rand-gossip(ρ, dline, dsize, α)
```

Algorithm 1: Main randomized continuous gossip routine at process i.

5.2 Randomized Algorithm Description

We now describe the randomized algorithm fixed-rand-gossip. The pseudocode can be found in Algorithm 2. The basic idea is as follows: The "participants" in each "instance" of fixed-rand-gossip repeatedly choose two random graphs defined by the edgesets N_a and N_b; the variable *dguess* controls the degree of these graphs. When the degree is sufficiently large, two things happen: first, the participants are connected in N_a by a small diameter graph, and hence they can rapidly exchange information on the edges of N_a; second, the participants are collectively connected to every process in the system via N_b, and hence can cooperatively deliver all their rumors. If there are too few participants, then *dguess* may never get sufficiently large, in which case each process simply sends its rumor directly to the specified destinations.

We now proceed to describe the pseudocode in more detail. The algorithm takes four parameters: a rumor ρ, a deadline *dline* that is no greater than $\rho.d$, a destination set size *dsize* that is at least as large as $|\rho.D|$, and a factor α controlling the growth of variable *dguess*. Consider some process i executing the fixed-rand-gossip routine in an attempt to distribute rumor ρ.

The goal of the main loop from lines 11–24 is to guess the right number of messages to send in each round, i.e., the right degree for the random graphs N_a and N_b. The loop terminates for a process i in two cases. Case 1: Process i learns that its rumor was successfully sent to every destination, in which case 1 is finished. The variable $sent[p]$ tracks which rumors have been sent to process p, and when $\rho \in sent[p]$ for every $p \in \rho.D$, then i can be sure that ρ has been successfully delivered (see line 23). Case 2: The variable *dguess* reaches, approximately, *dsize*. In the latter case, process i can abort the loop and simply send the rumor directly to the destinations $\rho.D$ (lines 25–28), as there are no more than *dsize* such destinations.

Recall that the variable *dguess* controls the degree of the random graph N_a, in which the participants communicate with each other, and the graph N_b, in which the participants communicate with the other processes. In both cases, the protocol succeeds in delivering rumors and terminating when *dguess* is sufficiently large (lines 12–22). That is, if there are k non-failed participants in these instances of fixed-rand-gossip, then when *dguess* is about $(n/k)\log^2 n$, the protocol completes.

We will later show that, in this case, the random graph N_a, when restricted to the participants, has a small (logarithmic) diameter; thus the participants can communicate efficiently with each other by exchanging messages along edges of N_a. (The graph N_a is selected in line 12, and lines 13–14 ensure that it is undirected.) Similarly, the subgraph of N_b induced by the participants has roughly $\Theta(n \log^2 n)$ outgoing random edges, i.e., enough to ensure that every possible destination in $[n]$ can be reached by one of the participants. Thus, once *dguess* is sufficiently large, the following steps take place: (1) after the first $O(\log n)$ rounds, all the participants learn about all the rumors; (2) in the next rounds, every rumor is sent to every destination; and (3) in the final $O(\log n)$ rounds, all the participants again exchange information and learn that their rumors have been delivered, allowing them to terminate. (Of course, this description is somewhat simplified, as participants may fail during the protocol.)

5.3 Algorithm Analysis

We analyze the algorithm assuming that $dline \geq 64$ and that $\alpha = \max(n^{1/\sqrt{dline}}, 2)$. The analysis also relies on the following fact, whose proof is relatively standard (see for example [2]).

FACT 2. *Given c, n, k, γ: Let $V = [n]$, and let $V' \subseteq V$ where $k = |V'|$. For each $v \in V'$, choose $16c(n/k)\gamma \log^2 n$ neighbors in V, resulting in an edgeset E. Then graph $G = (V', E)$ has diameter at most $\log_\gamma k$ with probability at least $1 - 1/n^c$.*

5.3.1 Analysis: Correctness.

We first argue that the rand-gossip protocol is correct, i.e., it delivers every admissible rumor by the deadline.

LEMMA 1. *The* rand-gossip *protocol delivers every admissible rumor by the specified deadline.*

PROOF SKETCH. First, it is easy to observe that every rumor is delivered, as, if it is not delivered, then the source sends it directly to the destinations (lines 25–28). It only remains to count the number of rounds, observing that $\left[\log_\alpha (dsize/\alpha^4)\right] \left[3 \cdot dline/(4 \log_\alpha (dsize)) + 1\right] + 1$ is always no greater than *dline*. □

5.3.2 Analysis: Message Complexity.

We now examine the message complexity of our randomized continuous gossip protocol. We first fix a round r, a deadline *dline*, and a size *dsize*, and analyze the messages sent by every invocation of fixed-rand-gossip$(\cdot, dline, dsize, \alpha)$. We also fix the set of rumors $\{\rho_1, \rho_2, \ldots, \rho_k\}$ that are associated with these invocations of fixed-rand-gossip. Define k to be the number of such rumors, and P to be the set of k processes where these rumors are injected.

The basic idea underlying the fixed-rand-gossip protocol is that as soon as *dguess* is sufficiently large to discover enough "collaborating" processes, then all the rumors injected at those processes are disseminated and the gossip protocol terminates. To that end, we begin by examining an iteration of lines 11–24, arguing that if $dguess \geq 16c(n/k)\log^2 n$, then

```
1  dguess : integer
2  N_a, N_b : set of neighbors
3  R : set of rumors
4  done : boolean
5  sent[] : array of sets of rumors, indexed by [n]
6  procedure fixed-rand-gossip(rumor ρ, integer dline, integer dsize, integer α)_i
7      R ← {ρ}
8      if (i ∈ ρ.D) then DELIVER(ρ)
9      dguess ← 1
10     done ← false
11     while (done = false) and (dguess ≤ dsize/α^4) do:
12         let N_a ← {n^{4 log_α(dsize)/dline} · dguess processes chosen uniformly at random from [n]}
13         send(NBR, i) to every process in N_a
14         for every (NBR, ℓ) message received : N_a ← N_a ∪ {ℓ}
15         repeat 3dline/(4 log_α(dsize)) rounds:
16             let N_b ← {dguess processes chosen uniformly at random from [n]}
17             send(RUMORS, R, sent) to every process in N_a ∪ N_b
18             for every p ∈ N_a ∪ N_b, for every rm ∈ R : sent[p] ← sent[p] ∪ {rm}
19             for every (RUMORS, R', sent') message received :
20                 for every ρ' ∈ R' : if (i ∈ ρ'.D) then DELIVER(ρ')
21                 R ← R ∪ R'
22                 for every p ∈ [n] : sent[p] ← sent[p] ∪ sent'[p]
23         if (∀p ∈ ρ.D : ρ ∈ sent[p]) then done ← true
24         dguess ← α · dguess
25     if (done = false) then
26         send(RUMORS, R) to every process in ρ.D
27         for every (RUMORS, R') message received :
28             for every ρ' ∈ R' : if (i ∈ ρ'.D) then DELIVER(ρ')
```

Algorithm 2: The fixed-rand-gossip routine at process i for specific values of *dline* and *dsize*.

the protocol completes. (Notice that since rounds are synchronous, processes that begin the gossip protocol in the same round also execute the protocol in lockstep, and hence processes execute the same line in the same round.)

LEMMA 2. *Consider a single iteration of lines 11–24 of* fixed-rand-gossip$(\cdot, dline, dsize, \alpha)$ *by processes in P. Let P' be the set of processes that: (i) do not set done = true prior to the beginning of the iteration, and (ii) do not fail by the end of the iteration. Let $k' = |P'|$. Let dguess$'$ be the value of dguess for every process in P at the beginning of the iteration. (Notice that every process in P that is not complete has the same value of dguess, as each increases it by the same factor α in each round.) If dguess$' \geq 16c(n/k')\log^2 n$, then every process in P completes or fails by the end of the iteration, with probability at least $1 - 1/n^{c-1}$.*

PROOF SKETCH. In each iteration, the subgraph of N_a induced by the non-failed participants has diameter at most $dline/(4\log_\alpha(dsize))$, by Fact 2. Thus, in the first iteration of lines 16–22, every participant learns every rumor. Next, observe that the subgraph of N_b induced by the non-failed participants has $\Theta(n \log n)$ outgoing edges leaving the subgraph, and hence every rumor is sent to every process, with high probability. Finally, in the last iteration of lines 16–22, every participant learns that all the rumors have been disseminated and then terminates. □

We can now analyze the overall performance, bounding the per-round message complexity:

LEMMA 3. *The per-round message complexity of* fixed-rand-gossip*, beginning in round r, associated with parameter dline and dsize, is at most $O\left(n^{1+5/\sqrt{dline}} \log^2 n\right)$, with probability at least $(1 - 1/n^{c-1})$.*

PROOF SKETCH. If any iteration of lines 11–24 satisfies the assumptions of Lemma 2, then it remains only to calculate the message complexity of that last iteration: we know that $dguess < 16c\alpha \cdot (n/k)\log^2 n$, for k and *dguess* defined according to Lemma 2, as otherwise the participants would have all completed in the previous iteration. This leads to the desired bound on message complexity.

On the other hand, if no iteration of lines 11–24 satisfies Lemma 2, then we again know that $dguess < 16c(n/k)\log^2 n$ throughout, leading to the same bound throughout. We bound the message complexity in the last round (lines 25–28) by also observing that $dguess > dsize/\alpha^5$, and hence we can bound $k \cdot dsize$, the maximum number of messages sent in the final round. □

5.3.3 Adaptive Per-Round Message Complexity

When there are relatively few rumors, or when there are relatively few destinations-per-rumor, it may be more efficient for each process to send its rumor directly to the destination, rather than participating in the gossip protocol. In order to obviate this scenario, we have designed the gossip protocol to always be as efficient, within $\log^{O(1)}(n)$ factors, as sending rumors directly. In particular, notice that *dguess* is bounded by *dsize*; thus no process ever sends more than *dsize* messages in a round.

LEMMA 4. *The per-round message complexity of the* fixed-rand-gossip *protocol beginning in round r associated*

with parameter dline and dsize is at most $O\left(\sum_{j=1}^{k} |\rho_j.D|\right)$ *(with probability 1).*

Lemmas 3 and 4 together yield our main result:

THEOREM 3. *The per-round message complexity of algorithm* rand-gossip *is at most* $O\left(\min\left\{n^{1+5\sqrt{2/d_m}}\log^6 n, \sum_{j=1}^{k}|\rho_j.D|\log^4 n\right\}\right)$, *with probability at least* $1-1/n^{c-2}$; d_m *is the minimum deadline of any rumor active in the system.*

6. DETERMINISTIC SOLUTION

In this section we show how to obtain a deterministic algorithm achieving message complexity only slightly larger than the randomized solution. For clarity of presentation, we focus on the case $dsize = n$ (thus automatically holding for all $dsize = O(n)$), and we do not address the issue of adaptavitiy. We call this algorithm det-gossip. We first present the basic theory behind expander graphs and their fault-tolerant properties; we then show how to deploy them in a similar scheme as seen in Section 5.

6.1 Fault-Tolerance of Expanders

Let G_a, for an integer $1 \leq a \leq n$, be a fixed regular a-expansion graph of n nodes, c.f., [22]. Formally, an *a-expander*, for a positive integer a, is defined as a simple graph with set of nodes V of size $n \geq a$ and such that if W_1 and W_2 are any sets of nodes, each of size at least a, then there is a node w_1 in W_1 and another node w_2 in W_2 such that $\{w_1, w_2\}$ is an edge in the graph. Let Δ_a stand for the maximum node degree of graph G_a.

The following result, due to Pinsker [21], describes dependency between expansion parameter a and the maximum node degree Δ_a (it can be proved using a standard probabilistic argument).

FACT 3. **[21]** *For any positive integers a and n, with $a \leq n$, there exists an a-expander with n nodes and of a maximum node degree $O((n/a)\log n)$.*

The best explicitly constructed a-expanders, for any integer a, are due to Ta-Shma, Umans and Zuckerman [24], who showed how to efficiently construct regular a-expanders with node degree $O((n/a)\log^{O(1)}(n))$.

We now prove a fault-tolerant property of a-expander graphs, stating that for sufficiently large number of non-faulty nodes there is a subset of at least a nodes that induce a subgraph of arbitrarily small diameter. Assume $\alpha \geq 2$ is an integer, and let $\beta = \log_{\alpha/2} a$. Consider a regular a-expander G. Let Q be a subset of nodes of size at least $\alpha \cdot a$, and let H denote the subgraph of G induced by nodes in Q. We use $N_H^j(W)$ to denote the set of nodes v such that v is of distance at most j in graph H from some node in W. Note that $N_H^{j+1}(W) = N_H(N_H^j(W))$, where $N_H(W)$ abbreviates $N_H^1(W)$. We also write $N_H^j(w)$, instead of $N_H^j(\{w\})$, if $W = \{w\}$ is a singleton.

LEMMA 5. *If a set $W \subseteq Q$ has size of at least a, then there exists a node $w \in W$ such that the set $N_H^\beta(w)$ is of a size larger than a.*

PROOF. The set $N_H(W) = N_G(W) \cap Q$ has size larger than $(\alpha/2)\cdot a$, since $|N_G(W)| > n - a$ and $|Q| \geq \alpha \cdot a \geq a + (\alpha/2)\cdot a$. Hence, by the pigeonhole principle, there is a set $W_1 \subseteq W$ of a size $\frac{a}{\alpha/2}$ such that $N_H(W_1)$ is of a size larger than a. We will extend this argument to show the existence of sets of nodes $W_j \subseteq W$ for $j = 1, \ldots, \beta$, with the following properties: (a) $|W_j| = \frac{a}{(\alpha/2)^j}$, and (b) $|N_H^j(W_j)| > a$.

This can be done by induction. The set W_1 has just been shown to exist. Suppose we have a set W_j with the required properties (a)-(b), for $j < \beta$. Observe that $|N_H^{j+1}(W_j)| > (\alpha/2)\cdot a$. This is because $N_H^{j+1}(W_j) = N_G(N_H^j(W_j)) \cap Q$, where set Q has size at least $\alpha \cdot a \geq a + (\alpha/2)\cdot a$, while set $N_G(N_H^j(W_j))$ has size bigger than $n - a$, due to expansion property and invariant (b) applied to $N_H^j(W_j)$. By the pigeonhole principle, there is a set $W_{j+1} \subseteq W_j$ such that $|W_{j+1}| = \frac{|W_j|}{\alpha/2}$ and the inequality $|N_H^{j+1}(W_{j+1})| > a$ holds. This completes the proof of the invariant. It implies that the set W_β is comprised of a single element $w \in W$, by definition of β, and set $N_H^\beta(w)$ has size larger than a. □

The following states the fault-tolerant property required by the expander graphs used in the deterministic algorithm presented in the next Section.

THEOREM 4. *Let G be an a-expander and $\alpha \geq 4$. For every set Q of at least $\alpha \cdot a$ nodes there is a subset $Q^* \subseteq Q$ of at least a nodes such that the subgraph of G induced by set Q^* has diameter of at most $2\beta = 2\log_{\alpha/2} a$.*

PROOF. Apply Lemma 5 to $W = Q$ and define Q^* as $N_H^\beta(w)$, where w is the node such that $|N_H^\beta(w)| > a$. By definition, every two nodes in Q^* are connected by a path of length at most 2β in graph G_a through nodes in Q^*, and in particular through node w. □

6.2 Deterministic Algorithm Description

In this section we present a description of det-gossip (Algorithm 3) and fixed-det-gossip (Algorithm 4). For similar reasons as in Section 5, we may assume that rumors were injected in the same round and they have approximately equal deadlines. Recall also that in this section we consider only destination sets of size n, i.e., broadcast requests; if they are smaller, algorithm det-gossip simply delivers the rumor to a larger set of all processes.

```
1 procedure det-gossip(rumor ρ)_i
2     dline ← min{21 log^3_{α/2} n, ρ.d}
3     dline ← 2^⌊log dline⌋
4     spawn fixed-det-gossip(ρ, dline, α)
```

Algorithm 3: Main deterministic continuous gossip routine at process i.

As in the case of the rand-gossip algorithm, the high-level det-gossip routine is simply a wrapper for the fixed-det-gossip routine: each deadline $\rho.d$ is rounded to the closest power of 2 that is larger than $\rho.d$ and no greater than $\Theta(\log^3_{\alpha/2} n)$, before invoking fixed-det-gossip; each fixed-det-gossip instance is invoked with three parameters: ρ, *dline* and α, where parameter α controls the growth of *dguess* and *dexcess* (its role and optimal value is slightly different from its randomized counterpart).

As in fixed-rand-gossip, the main idea underlying fixed-det-gossip is that all processes at which a rumor is injected cooperate to distribute the rumors. Instead of sending messages to random sets of processes, as in fixed-rand-gossip,

the processes communicate by sending/receiving messages from processes corresponding to its neighbors in carefully chosen expander graphs. By choosing these graphs with suitable expansion and node degree, we guarantee efficient delivery within the deadline. Let $G(x)$, for an integer $1 \leq x \leq n$ of the form α^j for some non-negative integer j, be a (n/x)-expander graph of maximum node degree $\Delta(x) = \Theta(x \log^{O(1)}(n))$. We also assume that $G(\alpha^j)$ is a subset of $G(\alpha^{j+1})$, as otherwise we could take a union of these two graphs and get a (n/α^{j+1})-expander of degree $\Theta(\alpha^{j+1} \log^{O(1)}(n))$ as required.

We refer to each iteration of the main loop of the algorithm as an **epoch** (lines 12–35). The main parameter associated with an epoch is *dguess*, which attempts to guess the size of the group of similar processes to be potential collaborators in rumor distribution. This parameter stays the same for the whole epoch, except its very last line 35 when it is increased by factor α.

The main part of an epoch is an inner loop (line 16), each iteration of which we refer to as a **stage** (lines 17–33). (Additional local computation within an epoch (lines 12-15 and 34-35) can be accounted to a single round of a stage.) In each stage, the variable *dexcess* is increased by a factor of α (line 33), increasing the degree of graph $G(\cdot)$ used to defining set N_b (c.f., line 28).

A stage consists of two parts, called *collaborating* and *working* periods. In the **collaborating period** (19–25), each process communicates $10 \log_{\alpha/2} n$ times with its neighbors in graph $G(dguess)$ having the same local round counter, epoch and stage number. The goal of this communication is to build two sets $Close\_Col \subseteq Col$ consisting of nodes with the same local round counter, epoch and stage number within distance at most $2 \log_{\alpha/2} n$ and $10 \log_{\alpha/2} n$, respectively, in graph $G(dguess)$; we often refer to these nodes as *close collaborators* and *collaborators*, respectively. Since collaborators must have been alive in the previous stage, the process may assume that they informed the processes they were supposed to inform in the previous stage, and thus it can update this knowledge in its progress set W (line 26). Nodes with sufficiently large sets of close collaborators will send messages in the succeeding working period, while the remaining ones will switch to the next epoch (c.f., line 27). Summarizing, collaborators are for recording the progress in rumor propagation, while close collaborators are to decide whether we have enough collaborators in the close neighborhood in $G(dguess)$ and may continue with the current epoch or we should switch to the next epoch instead, looking for more accurate set of (close-)collaborators (in graph $G(dguess \cdot \alpha)$).

The set W initially stores all the destinations for rumors, and is re-initialized in the beginning of each epoch (line 12); as processes are sent rumors, they are removed from W (see lines 26 and 31). In the second part of the stage, which we refer to as the **working period** (lines 27–31), each participant that has enough close collaborators (line 27) sends all the rumors it knows to processes in its set N_b, which consists of its neighbors in graph $G(dguess \cdot dexcess)$ that are still in set W (lines 28-29). The rumors are then delivered to the destinations and the progress in delivering rumors is recorded by the sender (lines 30-31). Even though the degree of graph $G(dguess \cdot dexcess)$ increases in every stage, the size of set N_b, and thus the number of RUMORS messages sent in the working period, does not necessarily grow substantially, since a process only sends messages to the neighbors that remain in W; actually, we show in the analysis that the amortized number of such messages per collaborator is bounded by $dguess \cdot \log^{O(1)}(n)$.

Finally, the last step of the routine is exactly as in fixed-rand-gossip: if a process cannot ensure that its rumor has been delivered, it simply sends its rumor directly to everyone (lines 36–38).

6.3 Algorithm Analysis

We first show that det-gossip guarantees QoD under any (worst-case adaptive) adversarial pattern of adversary $CRRI$, and then we analyze its per-round message complexity. For this purpose, we fix α such that $\alpha = n^{3/\sqrt[3]{dline}} \geq 4$ and $\Big((10 \log_{\alpha/2} n + 1) \cdot \log_\alpha n\Big) \log_\alpha n + 1 \leq dline$. We also assume that *dline* is sufficiently large, which for the purpose of our analysis means $dline \geq 100$.

In each round, we conceptually split processes into *groups* associated with a unique triple (t, g, x), corresponding to their values of $(rcount, dguess, dexcess)$. We say that a triple (t, g, x) is **valid** if it can be associated with some non-empty group of processes at some point of some execution. Let $X_{p,t}$ denote the value of a variable X in process p at the end of round t.

We start by showing that removing elements from set W is consistent with the progress of delivering the rumor of the removing process in its current epoch.

LEMMA 6. *If a process p removes the id of another process q from its set W during an epoch (lines 26 and 31) then the current rumor of p has been sent to process q in some round of the current epoch of process p.*

THEOREM 5. *Given any adversarial pattern $\mathcal{A} \in CRRI$, algorithm* det-gossip *guarantees QoD.*

PROOF SKETCH. Observe that a single run of routine fixed-det-gossip, when counting all the rounds, lasts at most $\Big((10 \log_{\alpha/2} n + 1) \cdot \log_\alpha n\Big) \log_\alpha n + 1 \leq dline$ rounds. If set W is emptied, then by Lemma 6 the rumor has been sent to all processes. Otherwise, it is sent directly to all processes in the last round while executing lines 37-38. □

We now proceed to analyze the message complexity of algorithm det-gossip. Consider a valid triple (t, g, x). We proceed by counting the number of messages sent in a round by processes associated with this triple. In particular, we will show that this is $O(\alpha^2 n \log^{O(1)}(n))$. Combining this with the upper bound $O(\log_\alpha^5 n)$ on the number of different valid triples, and with the bound $dline = O(\log_\alpha^3 n)$, we get the final result on the message complexity, c.f., Theorem 6. Let $Q(t, g, x)$ be the number of processes that are alive and are associated with triple (t, g, x) at the end of the considered (global) round of an execution, which is also round t of their routines fixed-det-gossip.

LEMMA 7. *There are no more than $\log_\alpha n \cdot \alpha^2 \cdot (n/g)$ processes associated with a triple (t, g, x) in a single round.*

PROOF SKETCH. If in some stage there are more than $\log_\alpha n \cdot \alpha^2 \cdot (n/g)$ processes associated with a triple, then in the previous epoch there must have been more than $\alpha^2 \cdot (n/g)$ of them associated with some triple $(t', g/\alpha, x')$, for any round counter t' of the previous epoch and stage x' of that epoch; yet if there were that many correct participants, they

```
1  rcount, dguess, dexcess, col_rounds: integer
2  N : set of neighbors
3  R : set of rumors
4  done, cont: boolean
5  Col, Close_Col, W: set of processes' ids
6  procedure fixed-det-gossip(rumor ρ, integer dline, integer α)_i
7      R ← {ρ}
8      rcount ← 1
9      dguess ← 1
10     done ← false
11     while (done = false) and (dguess ≤ n/α) do:                    // Iterating epochs
12         W ← [n] \ {i}
13         N_a ← the set of neighbors of node i in graph G(dguess)
14         dexcess ← 1
15         cont ← true
16         while (cont = true) and (dguess · dexcess ≤ n/α) do:        // Iterating stages
17             Col ← {i}
18             Close_Col ← {i}
19             for col_rounds ← 1 to 10 log_{α/2}(n) do:              // Lines 20-25: Collaborating period
20                 send(COLLABORATE, (rcount, dguess, dexcess), R, Col, Close_Col) to every process in N_a
21                 for every (COLLABORATE, (t, g, x), R', Col', Close_Col') message received and such that
                   (t, g, x) = (rcount, dguess, dexcess) :
22                     R ← R ∪ R' and DELIVER(R')
23                     Col ← Col ∪ Col'
24                     if col_rounds > 8 log_{α/2}(n) then Close_Col ← Close_Col ∪ Close_Col'
25                 rcount ← rcount + 1
26             if dexcess ≥ α then W ← W \ {neighbors of nodes in Col in graph G(dguess · dexcess/α)}
27             if |Close_Col| < n/dguess then (cont ← false) else     // Lines 27-31: Working period
28                 N_b ← the set of neighbors of node i in graph G(dguess · dexcess) intersected with W
29                 send(RUMORS, R) to every process p in N_b
30                 for every (RUMORS, R') message received : DELIVER(R')
31                 W ← W \ N_b
32             rcount ← rcount + 1
33             dexcess ← α · dexcess
34         if W = ∅ then done ← true
35         dguess ← α · dguess
36     if (done = false) then
37         send(RUMORS, R) to every process
38         for every (RUMORS, R') message received : DELIVER(R')
```

Algorithm 4: The deterministic routine fixed-det-gossip at process i for a specific value of $dline$.

would have collaborated in the previous epoch to distribute the rumors, by Theorem 4, instead of changing it (line 27); this leads to contradiction. □

We now consider the progress in propagating rumors during the working period — the last round of a stage. This is important for bounding the number of RUMORS messages sent by processes with the same triple (t, g, x).

LEMMA 8. *Consider triple (t, g, x) such that $x \geq \alpha$ and all processes in $Q(t, g, x)$ execute line 27 during their round with counter t. The union U of sets W taken over processes p in $Q(t, g, x)$ satisfying $|Close\_Col_{p,t}| \geq n/g$ (i.e., processes p that send some* RUMORS *messages in the working period, during the succeeding line 29), taken before sending any* RUMORS *message in line 29, is of size smaller than $\alpha \cdot n/(g \cdot x)$.*

PROOF SKETCH. Consider nodes p, q sending messages in the working period. Their sets of close collaborators must be of size at least n/g. By $\alpha \cdot n/(g \cdot x)$-expansion of graph $G(g \cdot x/\alpha)$ used in the working period of the previous stage, there is an edge between sets of close collaborators of p and q in this graph. Therefore through this edge these two sets exchange information about themselves during the collaborating period in-between the considered working periods. Each of these sets of close collaborators is large enough to have more than $n - \alpha \cdot n/(g \cdot x)$ neighbors in graph $G(g \cdot x/\alpha)$ used in the preceding working period, which assures that before the next working period sets W of processes p, q shrink to less than $\alpha \cdot n/(g \cdot x)$ elements each. □

We are now ready to bound the message complexity of algorithm det-gossip.

THEOREM 6. *The per-round message complexity of algorithm* det-gossip *is at most $O(n^{1+6/\sqrt[3]{d_m}} \log^{O(1)}(n))$, d_m being the minimum deadline of any rumor active in the system.*

PROOF SKETCH. We need to bound each of the two types of messages for each triple (t, g, x). Lemma 7 bounds the number of COLLABORATE messages to $O(\log_\alpha(n) \cdot \alpha^2 \cdot (n/g) \cdot \Delta(g))$, since there are never too many collaborators. The RUMORS messages are bounded by Lemma 7 to $O(\log_\alpha n \cdot \alpha^2 \cdot n/(g \cdot x) \cdot \Delta(g \cdot x))$. The bound follows from the

observation that there are at most $\log_\alpha^5 n$ valid tuples, and by the estimates $\alpha = n^{3/\sqrt[3]{d_m}}$ and $\Delta(a) = a \log^{O(1)}(n)$. $\square$

7. CONCLUSION AND FUTURE WORK

In this paper we have introduced and studied a novel version of the gossip problem, called Continuous Gossip, where n crash-and-restart-prone processes are continuously subject to injected rumors. Rumors have deadlines and the goal is for all rumors, subject to a condition that we call quality-of-delivery (QoD), to be disseminated. We first presented lower bounds on the per-round message complexity of randomized and deterministic continuous gossip and then we described and analyzed an efficient (close to optimal) randomized algorithm and its de-randomized version that guarantee QoD in every execution. We note that our algorithms can be implemented in such a way that each message sent carries only rumors and at most $O(n^2)$ additional bits. (Hence if the rumors are large, the per-round bit complexity of the algorithms is essentially the per-round message complexity times the size of the rumors.) It would be interesting to investigate whether the bit complexity of the algorithms can be reduced even further.

Several future research directions emanate from this work. For example, it would be interesting to investigate whether it is possible to close the gap between our presented lower and upper bound results. Also, there is the question of whether it is possible to obtain subquadratic per-round message complexity for rumors with very short deadlines, i.e., $d < 64$ for randomized and $d < 100$ for deterministic gossip.

Other interesting future directions involve variants of the model we have introduced. For example, perhaps, other types of QoD could be considered. Along with message complexity, we could also consider latency as an efficiency measure, especially for long deadlines (e.g., of duration $\Omega(\log^{O(1)}(n))$); it would be interesting to study the trade-offs between deadlines, message complexity and latency. An even more challenging research direction would be to consider partially-synchronous continuous gossip, perhaps in an analogous manner as the one-shot n-rumor gossip problem was considered in [13]. Last but not least, we would like to use our continuous gossip algorithms as a building block in improving the communication cost of other related problems such as "continuous" consensus and "continuous" cooperative task computing.

8. REFERENCES

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

[2] B. Bollobas and W. Fernandez de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2): 125–134, 1982.

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

[4] M.R. Capalbo, O. Reingold, S.P. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *STOC 2002,* pages 659–668.

[5] B. S. Chlebus and D. R. Kowalski. Robust gossiping with an application to consensus. *J. Comput. Syst. Sci.,* 72(8): 1262–1281, 2006.

[6] B. S. Chlebus and D. R. Kowalski. Time and communication efficient consensus for crash failures. In *DISC 2006,* pages 314–328.

[7] B. S. Chlebus, D. R. Kowalski, and A. A. Shvartsman. Collective asynchronous reading with polylogarithmic worst-case overhead. In *STOC 2004,* pages 321–330.

[8] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC 1987,* pages 1–12.

[9] K. Diks and A. Pelc. Optimal adaptive broadcasting with a bounded fraction of faulty nodes. *Algorithmica,* 28(1): 37–50, 2000.

[10] B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading: Expanders, push vs pull, and robustness. In *ICALP 2009,* pages 366–377.

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

[12] Z. Galil, A. Mayer, and M. Yung. Resolving message complexity of Byzantine agreement and beyond. In *FOCS 1995,* pages 724–733.

[13] C. Georgiou, S. Gilbert, R. Guerraoui, and D.R. Kowalski. On the complexity of asynchronous gossip. In *PODC 2008,* pages 135–144.

[14] Ch. Georgiou, D. R. Kowalski, and A. A. Shvartsman. Efficient gossip and robust distributed computation. *Theor. Comput. Sci.,* 347(1-2): 130–166, 2005.

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

[16] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized Rumor Spreading. In *FOCS 2000,* pages 565–574.

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

[18] A. Kermarrec, L. Massoulie, and A. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. on Parallel and Distr. Syst.,* 14(3): 248–258, 2003.

[19] D. R. Kowalski and M. Strojnowski. On the communication surplus incurred by faulty processors. In *DISC 2007,* pages 328–342.

[20] A. Pelc. Fault-tolerant broadcasting and gossiping in communication networks. *Networks,* 28: 143-156, 1996.

[21] M.S. Pinsker. On the complexity of a concentrator. In *Proc. of 7th Annual Teletraffic Conference,* 1973.

[22] N. Pippenger. Sorting and selecting in rounds. *SIAM J. Computing,* 16: 1032–1038, 1987.

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

[24] A. Ta-Shma, C. Umans, and D. Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *STOC 2001,* pages 143–152.