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

Chryssis Georgiou[*] Seth Gilbert[†] Dariusz R. Kowalski[‡]

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.

Keywords: Gossip, crashes and restarts, dynamic rumor injection, random and expander graphs.

[*] Dept. of Computer Science, University of Cyprus, CY-1048 Nicosia, Cyprus. Email: chryssis@cs.ucy.ac.cy

[†] Dept. of Computer Science, National University of Singapore, Singapore 119077. Email: seth.gilbert@comp.nus.edu.sg

[‡] Dept. of Computer Science, University of Liverpool, Liverpool L69 3BX, UK. Email: D.Kowalski@liverpool.ac.uk

1 Introduction

Disseminating information lies at the core of distributed computing. *Gossiping* is a fundamental mechanism for achieving efficient data dissemination [21]. 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 [6, 13, 14], database consistency [9], cooperative task computing [15], failure detection [24], group communication [2, 12, 19], and resource location [18].

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.

Of course, the obvious solution to *continuous* gossip might seem to be repeatedly running instances of classical, *static* gossip. However, one runs into a variety of problems. First, it is hard for processes to synchronize their instances of static gossip, as processes crash and restart with no knowledge of the global time (and agreeing on the time is non-trivial in such a model). This does not necessarily lead to an efficient solution for a variety of reasons. First, processes need to synchronize their static gossip procedures; if processes do not have access to a common global clock (which is non-trivial to implement in a crash-restart model) this approach may not succeed. In addition, rumors may have different sized destination sets, while static gossip tends to deliver rumors to everyone.

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, [16, 20, 21].) 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 [11, 17, 18, 19]). 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 [17] 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 [7]. 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 [20], 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 [21] together with the book by Hromkovic et al. [16] 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 [7] and Kowalski and Strojnowski [20] 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 [5, 6, 7, 10, 15] and shared memory [8]. In this work we study a new fault-tolerant property of expander graphs, in the context of a trade-off between the initial number of non-faulty nodes and the diameter of the connected component induced by some large subset of non-faulty nodes.

The *collect problem* [25] can be considered the counter-part of the gossip problem in the shared-memory model: n processes in a shared-memory system must each learn the values of n registers. Although the goal is the same, the perspective and the techniques involved are rather different mainly due to the different communication model (see for example [25, 1, 8]).

Paper organization. In Section 2 we present a model of computation, and in Section 3 we formulate the notion of quality of delivery and define the Continuous Gossip problem. In Section 4 we present lower bounds on the per-round message complexity for randomized and deterministic continuous gossip. In Section 5 we present an efficient randomized algorithm for continuous gossip. In Section 6 we show how expander graphs can be used to derandomize our randomized algorithm. We conclude in Section 7.

2 Model

In this section we present the model of computation considered in this work.

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 say that rumor ρ is **active** in round t if it was injected no later than round t and has a deadline after or during 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). This implies that failures and rumor injections are independent of the random choices made during the execution. (For deterministic algorithms, there is no distinction between adaptive and oblivious adversary, as the adversary knows a priori how a deterministic algorithm would behave under a specific adversarial pattern.)

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 (i.e., $1 - 1/n^{\Omega(1)}$), $M_t(Rand, \mathcal{A})$ is at most $M(Rand)$. (We specify the precise probability when analyzing such algorithms.)

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 3.1 *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. For example, if we assume that the adversary injects only rumors that need to be sent to all processes in the system, then the above naive protocol may have per-round message complexity of $\Theta(n^2)$.

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 in *any execution*.

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)[1]. We first show a lower bound for deterministic and then a lower bound for randomized algorithms.

Theorem 4.1 *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* $M(Det) = \Omega(n^{1+1/d}/d)$.

Proof: If $d > n/4$, it is obvious that the message complexity must be $\Omega(1)$, and hence the bound holds. Assume $d \leq n/4$. Consider a continuous gossip algorithm Det, and let M denote an upper bound on the per-round message complexity of the algorithm. Our goal is to show that $M \geq (n/2)^{1+1/d}/d$. Consider the following adversarial strategy:

In the first round, a single rumor is injected with deadline d at each process. In each of the following d rounds, the adversary crashes each process that receives at least $(n/2)^{1/d}$ messages in a round. The adversary does not restart any processes in these rounds. It follows that the number of crashed processes in the considered part of the execution is at most $\frac{M}{(n/2)^{1/d}} \cdot d$.

A straightforward induction argument shows that since each non-crashed process receives less than $(n/2)^{1/d}$ messages at any round j, for $2 \leq j \leq d+1$, it knows less than $\prod_{j=2}^{d+1}(n/2)^{1/d} = (n/2)^{d \cdot 1/d} \leq n/2$ rumors by the end of round $d+1$. It follows that the number of processes crashed during the first $d+1$ rounds must be at least $n - n/2 = n/2$; otherwise QoD is not guaranteed, as each non-crashed process does not collect at least $n/2$ admissible rumors. Hence, we get $\frac{M \cdot d}{(n/2)^{1/d}} \geq n/2$, which implies $M \geq (n/2)^{1+1/d}/d$. □

Theorem 4.2 *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* $M(Rand) = \Omega(n^{1+1/d}/d^2)$.

[1] Although these "restricted" adversarial patterns conduce a special case of the general Continuous Gossip problem defined in the previous section, the derived lower bounds hold for the general Continuous Gossip problem.

Proof: We show that $M \geq (n/2)^{1+1/d}/(4d^2)$. As in the proof of Theorem 4.1 we may assume that $d \leq n/4$. Since adversary *CRRI* is oblivious, it needs to perform a simulation of the algorithm in order to specify its adversarial pattern before the real computation starts. It knows that it will inject a single rumor at each process in the first round. The simulation proceeds for the subsequent d rounds, and for the currently considered round it computes an expected number of received messages for each non-crashed process, under fixed behavior in the previous rounds. Then, the adversary crashes those that are to receive at least $(n/4)^{1/d}/(2d)$ messages in expectation. The number of crashed processes in a round is therefore at most $\frac{M}{(n/2)^{1/d}/(2d)}$ with probability bigger than $1 - 1/(2d)$.

It can be easily shown that this can be continued throughout the considered d rounds, and finally gives at most $\frac{M}{(n/2)^{1/d}/(2d)} \cdot d = \frac{M \cdot 2d2}{(n/2)^{1/d}}$ crashes with probability bigger than $1 - d \cdot 1/(2d) = 1/2$. As in the analysis for Theorem 4.1, a single non-crashed process at the end of round $d+1$ knows less than $\prod_{j=2}^{d+1} (n/2)^{1/d} = (n/2)^{d \cdot 1/d} \leq n/2$ other rumors, with probability bigger than $1 - d \cdot 1/(2d) = 1/2$. Therefore, by the probabilistic method, there is a determined execution $\mathcal{E}$ with the following two properties at the end of round $d+1$: the total number of crashed processes is at most $\frac{M \cdot 4d^2}{(n/2)^{1/d}}$, and there is a non-crashed process knowing less than $n/2$ rumors. Therefore, $\frac{M \cdot 4d^2}{(n/2)^{1/d}}$ must be at least $n - n/2 = n/2$, to guarantee QoD for rumors of non-crashed processes, which implies $M \geq (n/2)^{1+1/d}/(4d^2)$. □

From Theorems 4.1 and 4.2 it can be observed that the per-round message complexity of randomized algorithms can be slightly smaller than in the case of deterministic algorithms, leaving open the possibility that a randomized algorithm could perform slightly better than a deterministic algorithm. On the other hand, the fact that the gap between the lower bounds is relatively small could suggest that randomization does not help much.

Also 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. We begin in Section 5.1 with an overview of our solution. In Section 5.2, we present the rand-gossip protocol. In Section 5.3, we show that rand-gossip achieves $O(n^{1+5\sqrt{2/d_m}} \log^{O(1)}(n))$ per-round message complexity, with high probability, where d_m is the minimum deadline of any rumor active in the system. We then discuss the issue of adaptivity, and show that the per-round message complexity of rand-gossip is adaptive to the number of active rumors and their destinations.

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

```
procedure rand-gossip(rumor $\rho$)$_i$
    $dline \leftarrow \min\{25\log^2 n, \rho.d\}$
    $dline \leftarrow 2^{\lfloor \log dline \rfloor}$
    $dsize \leftarrow 2^{\lceil \log |\rho.D| \rceil}$
    spawn fixed-rand-gossip($\rho$, $dline$, $dsize$, $\alpha$)
```

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

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[2].

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 5.6.

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

[2] Some log-factor improvements can be gained by avoiding this partitioning, but the resulting protocol is significantly more complicated.

```
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 $\rho$, integer $dline$, integer $dsize$, integer $\alpha$)$_i$
7      $R \leftarrow \{\rho\}$
8      if $(i \in \rho.D)$ then DELIVER$(\rho)$
9      $dguess \leftarrow 1$
10     $done \leftarrow$ false
11     while $(done =$ false$)$ and $(dguess \leq dsize/\alpha^4)$ do:
12         let $N_a \leftarrow \{n^{4\log_\alpha(dsize)/dline} \cdot dguess$ processes chosen uniformly at random from $[n]\}$
13         send(NBR, $i$) to every process in $N_a$
14         for every (NBR, $\ell$) message received : $N_a \leftarrow N_a \cup \{\ell\}$
15         repeat $3dline/(4\log_\alpha(dsize))$ rounds:
16             let $N_b \leftarrow \{dguess$ processes chosen uniformly at random from $[n]\}$
17             send(RUMORS, $R$, $sent$) to every process in $N_a \cup N_b$
18             for every $p \in N_a \cup N_b$, for every $rm \in R$ : $sent[p] \leftarrow sent[p] \cup \{rm\}$
19             for every (RUMORS, $R'$, $sent'$) message received :
20                 for every $\rho' \in R'$ : if $(i \in \rho'.D)$ then DELIVER$(\rho')$
21                 $R \leftarrow R \cup R'$
22                 for every $p \in [n]$ : $sent[p] \leftarrow sent[p] \cup sent'[p]$
23         if $(\forall p \in \rho.D : \rho \in sent[p])$ then $done \leftarrow$ true
24         $dguess \leftarrow \alpha \cdot dguess$
25     if $(done =$ false$)$ then
26         send(RUMORS, $R$) to every process in $\rho.D$
27         for every (RUMORS, $R'$) message received :
28             for every $\rho' \in R'$ : if $(i \in \rho'.D)$ then DELIVER$(\rho')$
```

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

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 (line 16) 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 (lines 12–23): (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 (line 23), 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 [3]) and can be found in the appendix.

Fact 5.1 *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.

Theorem 5.2 *The* rand-gossip *protocol delivers every admissible rumor by the specified deadline.*

Proof: Fix some rumor ρ that is injected in round t at process q; recall from Algorithm 1 that $dline$ is set equal to $2^{\lfloor \log \rho.d \rfloor}$ and $dsize$ to $2^{\lceil \log |\rho.D| \rceil}$. We analyze the execution of fixed-rand-gossip$(\rho, dline, dsize, \alpha)$ that begins in round t. Assume that process q does not fail until after the deadline, as otherwise rumor ρ is not admissible for any $\ell \in \rho.D$. This implies that process q remains alive until at least round $t + dline$.

We will first show that every rumor is eventually delivered. Observe that either process q learns that ρ has been sent to every destination (line 23), or it sends the rumor ρ directly to every destination (lines 25–28). We need to argue that if process q skips lines 25–28, it is the case that indeed rumor ρ has been sent to all processes in $\rho.D$. For process q to skip executing lines 25–28 it means that q has finished its execution (wrt rumor ρ) with variable $done$ set to $true$ (line 25). It follows from line 23 that $\forall p \in \rho.D,\ \rho \in sent_q[p]$. For contradiction, assume that for a process $z \in \rho.D$, $\rho \in sent_q[z]$, but no process has sent ρ to z. First observe that $\rho \in R_q$ (line 7) and no rumor is ever removed from any R or $sent$ (for the given execution of fixed-rand-gossip). Now observe that q would include ρ in $sent_q[z]$ in two cases:

- Process q sends ρ to z. This happens if z is in $N_a \cup N_b$ in some round (lines 17–18). So assume this is not the case (otherwise we are done).
- Process q is informed by another process that ρ was sent to z (lines 19–22). In particular, q receives a message from some process w in which $\rho \in sent_w[z]$, and hence, per line 22, ρ will be included in $sent_q[z]$. By a simple reverse inductive argument, it follows that w has indeed sent ρ to z or it was informed by another process that ρ was sent to z (the reverse induction eventually ends at the first bullet above — z learns about ρ from a neighboring process that is aware of ρ). Otherwise we reach a contradiction to the fact that in our model of computation messages are not fabricated or processes do not exhibit a byzantine behavior (in which case a process could lie of sending a rumor to another process).

Therefore, every rumor is indeed eventually delivered. It now remains to ensure that every rumor is sent by round $t + dline$. Notice that the main loop (lines 11-24) executes at most $\log_\alpha(dsize)$ times, and each iteration takes $3dline/4\log_\alpha(dsize) + 1$ rounds (i.e., one round for lines 13–14, and the remaining rounds for the loop on lines 15–22). Lines 25–28 require one additional round. Thus, the total number of rounds is:

$$\log_\alpha(dsize) \cdot \left(\frac{3dline}{4 \log_\alpha(dsize)} + 1 \right) + 1$$

Recall that $\alpha \geq n^{1/\sqrt{dline}}$, and also that $dsize \leq n$. This implies that:

$$\log_\alpha(dsize) \quad \leq \quad \frac{\log dsize}{\log \alpha} \leq \frac{\sqrt{dline} \log dsize}{\log n} \leq \sqrt{dline}$$

We also observe that since size $dline \geq 64$, we conclude (easily) that $4 \leq 4\sqrt{dline}$ and that $8\sqrt{dline} \leq dline$. We then bound the number of rounds as follows:

$$\begin{aligned} \log_\alpha(dsize) \cdot \left(\frac{3dline}{4 \log_\alpha(dsize)} + 1 \right) + 1 \quad &\leq \quad \frac{3dline + 4\log_\alpha(dsize) + 4}{4} \leq \frac{3dline + 4\sqrt{dline} + 4}{4} \\ &\leq \quad \frac{3dline + 8\sqrt{dline}}{4} \leq \frac{3dline + dline}{4} \leq dline \end{aligned}$$

Thus rumor ρ is delivered by round $t + dline$, as desired. □

5.3.2 Analysis: Message Complexity

We now examine the message complexity of our randomized continuous gossip protocol. We first fix a round t, 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 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 5.3 *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: Consider the (undirected) graph among nodes in P' induced by the neighbor set N_a selected in line 12. We refer to this graph as $G = (P', N_a)$. Notice that this graph G is formed by each node in P' choosing $n^{4\log_\alpha(dsize)/dline} \cdot dguess'_r$ neighbors in $[n]$, i.e., at least $n^{4\log_\alpha(dsize)/dline} \cdot 16c(n/k')\log^2 n$

neighbors in $[n]$ (by assumption on the size of $dguess'$). By Fact 5.1, we conclude that this graph has diameter at most $\log(k') dline/4\log(n)\log_\alpha(dsize)$. Since $k' \leq n$, this implies that the diameter of G is at most $dline/4\log_\alpha(dsize)$.

Notice that lines 16–22 are repeated $3dline/4\log_\alpha(dsize)$, i.e., sufficiently frequently for information to propagate the diameter of the graph G at least 3 times. Fix an arbitrary rumor ρ' that was injected at a process in P'. Whenever a process p_j that has rumor $\rho' \in R$ sends a message, all the neighbors of p_j in G learn ρ'. We thus conclude that by the end of the first $dline/4\log_\alpha(dsize)$ iterations of lines 16–22, every process in P' has received rumor ρ'; more generally, every rumor injected at a process in P' is known by every other process in P' at this point.

Next, consider the first iteration of lines 16–22 after every rumor injected at a process in P' is known by every other process in P' at this point. We consider the sets N_b chosen by the processes in P'. Notice that, collectively, the processes in P' choose $k' \cdot dguess'$ random processes in $[n]$. By assumption, $k' \cdot dguess' \geq 16cn\log^2 n$. Thus, with probability at least $(1 - 1/n^c)$, every process in $[n]$ is chosen by at least one process in P'. Thus, within one round, every rumor injected at a process in P' has been sent to every process in $[n]$, with probability at least $1 - 1/n^c$. (Recall that none of the process in P' fail by the end of the iteration.) Specifically, for every rumor ρ' injected at a process in P', and for every process $p_\ell \in [n]$, there is some process p_j that has $\rho \in sent[p_\ell]$.

Finally, in the next $dline/4\log_\alpha(dsize)$ of lines 16–22, the information regarding which rumors were sent to which processes is propagated to every process in P'. As such, by the end of the last iteration of lines 16–22, for every process $p_j \in P'$ with rumor ρ', for every $p_\ell \in [n]$, rumor $\rho' \in sent[p_\ell]$ of p_ℓ. Thus, in line 23, process p_j sets $done = \mathsf{true}$, completing the protocol. Thus by the end of lines 11–24, every process in P has either completed or failed. By a union bound, the two probabilistic events (i.e., the small diameter of the graph and the distribution in lines 16–22) occur with probability at least $(1 - 1/n^c - 1/n^c) \geq 1 - 1/n^{c-1}$. □

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

Lemma 5.4 *The per-round message complexity of* fixed-rand-gossip, *beginning in round t, 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: There are two cases to consider, depending on whether any iteration of lines 11–24 satisfy the assumptions of Lemma 5.3. Assume first that there is some iteration of lines 11–24 where:

- Let P' be the set of processes that: (i) do not set $done = \mathsf{true}$ 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;
- $k' \cdot dguess' \geq 16cn\log^2 n$.

Then, by Lemma 5.3, every process completes by the end of the iteration, with probability at least $(1 - 1/n^{c-1})$. Moreover, the maximum per-round message complexity is achieved in this final iteration; we now bound this message complexity.

Since this is the first iteration satisfying the assumptions of Lemma 5.3, we can conclude that in the immediately preceding iteration, if k'' and $dguess''$ are defined as expected, then $k'' \cdot dguess'' < 16cn\log^2 n$. Since $dguess$ is incremented by a factor of α in each iteration, and since $k' \leq k''$, we conclude that in the final iteration, $dguess' < 16c\alpha n \log^2 n/k'$. Thus, the number of messages sent in each round during the final iteration is:

$$\begin{aligned} k'(|N_a| + |N_b|) &\leq O\left(k' \cdot n^{4\log_\alpha(dsize)/dline} dguess'\right) \leq O\left(n^{4\sqrt{dline}/dline} \cdot 16c\alpha n \log^2 n\right) \\ &\leq O\left(\left(2 + n^{1/\sqrt{dline}}\right) n^{1+4/\sqrt{dline}} \log^2 n\right) \leq O\left(n^{1+5/\sqrt{dline}} \log^2 n\right) \ . \end{aligned}$$

Since there are no messages sent after this final iteration, this concludes the first case, yielding the desired per-round message complexity.

We now focus on the second case: in every iteration of lines 11–24, $k' \cdot dguess' < 16cn\log^2 n$ (where k' and $dguess$ are defined as described in the statement of Lemma 5.3). Thus, for each iteration, we can bound the per-round message complexity of lines 11–24 exactly as above:

$$\begin{aligned} k'(|N_a| + |N_b|) &= O\left(k' \cdot n^{4\log_\alpha(dsize)/dline} dguess'\right) \leq O\left(n^{4\sqrt{dline}/dline} \cdot 16cn \log^2 n\right) \\ &\leq O\left(n^{1+4/\sqrt{dline}} \log^2 n\right) \end{aligned}$$

It remains to bound the per-round message complexity of lines 25–28. (Unlike the previous case, there may be processes that do not complete during iterations of lines 11–24.) Each process that has not yet completed sends $dsize$ messages, and hence if k' is the number of processes remaining at the beginning of the final iteration of lines 11–24, then the per-round messages complexity of lines 25–28 is at most $k' dsize$.

We have also assumed, for this case, that in all iterations (and, in particular, in the final iteration) of lines 11–24: $k' \cdot dguess' < 16cn\log^2 n$. In addition, in the final iteration, $dguess' \geq (dsize/\alpha^4)/\alpha$; otherwise after increasing $dguess'$ by a factor of α, there would be another iteration of lines 11–24. Putting these two facts together, we get that:

$$k' \cdot dsize \leq \frac{16cn\log^2 n \cdot dsize}{dguess'} \leq \frac{16cn\log^2 n}{1/\alpha^5} \leq 16c(2 + n^{5/\sqrt{dline}})n\log^2 n \leq O\left(n^{1+5/\sqrt{dline}} \log^2 n\right) .$$

This completes the proof. □

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 5.5 *The per-round message complexity of the* fixed-rand-gossip *protocol beginning in round t associated with parameter dline and dsize is at most* $O\left(\sum_{j=1}^{k} |\rho_j.D|\right)$ *(with probability 1).*

Proof: Fix a specific rumor ρ_j injected at a process p_j. We examine the messages sent by process p_j. First, observe that on lines 25–28, process p_j sends at most $|\rho_j.D|$ messages. It remains to bound the number of messages sent in each round by lines 11–24.

For each iteration of the loop from lines 11–24, process p_j sends $(|N_a| + |N_b|)$ messages per round. Since (as was shown in the proof of Theorem 5.2) $\log_\alpha(dsize) \leq \sqrt{dline}$, the per round message complexity for process p_j is bounded by:

$$\begin{aligned}(|N_a| + |N_b|) &\leq dguess(n^{4\log_\alpha(dsize)/dline}) + dguess \leq (2dsize/\alpha^4)(n^{4\log_\alpha(dsize)/dline}) \\ &\leq (2dsize/n^{4/\sqrt{dline}})n^{4/\sqrt{dline}} \leq 4|\rho_j.D|\end{aligned}$$

Summing over all rumors at all processes $p_j \in P$ yields the desired bound. □

Lemmas 5.4 and 5.5 together yield our main result.

Theorem 5.6 *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.

Proof: In any given round, there are at most $\log(dsize) \cdot \log(dline)$ different instances of fixed-rand-gossip being initiated. Each instance of fixed-rand-gossip lasts at most $25\log^2 n$ rounds (as we truncate each deadline after this point). Thus, in any given round, there are at most $O(\log^4 n)$ ongoing instances of fixed-rand-gossip. The total message complexity in a given round for rand-gossip is the sum of the per-round message complexities of these instances.

For a given round t, let $R_1, \ldots, R_{\log^4 n}$ be the sets of rumors associated with each of the $O(\log^4 n)$ instances of fixed-rand-gossip. By Lemmas 5.4 and 5.5, each instance induces the minimum of $O\left(n^{1+5\sqrt{2/d_m}}\log^2 n\right)$ messages and $\sum_{\rho\in R_\ell}|\rho.D|$ messages. (Recall that for every rumor ρ, the calculated deadline $dline > \rho.d/2$, and $\rho.d/2 \geq d_m/2$ by definition.)

Consider the instance ℓ of fixed-rand-gossip that generate the largest number of messages in round t. Clearly, the total number of messages generated by the $O(\log^4 n)$ instances is the minimum of $O\left(n^{1+5\sqrt{2/d_m}}\log^6 n\right)$ messages and $\sum_{\rho\in R_\ell}|\rho.D|\log^4 n$ messages, for some instance ℓ. Since $\sum_{\rho\in R_\ell}|\rho.D| \leq \sum_{\rho\in\bigcup R_\ell}|\rho.D|$, the result follows. □

Remark 5.7 *If we restrict adversary CRRI in adversarial patterns in which all rumors must be sent to all processes in the system (that is, for every rumor ρ, $\rho.D = [n]$) then it is easy to see that the* rand-gossip *algorithm has per-round message complexity of $O(n^{1+5/\sqrt{dline}}\log^{O(1)}(n))$. When this is contrasted with the lower bound result of Theorem 4.2 one can conclude that algorithm* rand-gossip *is close to optimal. In fact, for $dline = \Theta(\log^{O(1)}(n))$ algorithm* rand-gossip *is optimal within log factors.*

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 adaptivity. We call this algorithm det-gossip. We first present the basic theory behind expander graphs and prove their fault-tolerant properties; we then show how to deploy them in a similar scheme as seen in Section 5. Note that this de-randomization is not straightforward, as apart of using expander graphs with specific fault-tolerant properties, additional mechanisms assuring sufficient level of deterministic synchronization and collaboration must be introduced.

6.1 Fault-Tolerance of Expanders

Let G_a, for an integer $1 \le a \le n$, be a fixed regular a-expansion graph of n nodes, c.f., [23]. Formally, an *a-expander*, for a positive integer a, is defined as a simple graph with set of nodes V of size $n \ge a$ 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 [22], describes dependency between expansion parameter a and the maximum node degree Δ_a (it can be proved using a standard probabilistic argument).

Fact 6.1 [22] *For any positive integers a and n, with $a \le n$, 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 [26], 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 $\alpha \ge 2$ 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 6.2 *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 $|Q| \ge \alpha \cdot a \ge a + (\alpha/2) \cdot a$. 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 $\alpha \cdot a \ge a + (\alpha/2) \cdot a$, 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 6.3 *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 6.2 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.

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. An additional challenge is to synchronize activities of processes; for example, different processes may be using different expander graphs at the same time in the algorithm, as we do not assume a global clock. Such synchronization also allows to recognize quickly if a process crashes and restarts in a short period of time.

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.

```
1  procedure det-gossip(rumor $\rho$)$_i$
2      $dline \leftarrow \min\{21 \log^3_{\alpha/2} n, \rho.d\}$
3      $dline \leftarrow 2^{\lfloor \log dline \rfloor}$
4      spawn fixed-det-gossip($\rho, dline, \alpha$)
```

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

```
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 $\rho$, integer $dline$, integer $\alpha$)$_i$
7      $R \leftarrow \{\rho\}$
8      $rcount \leftarrow 1$
9      $dguess \leftarrow 1$
10     $done \leftarrow$ false
11     while ($done =$ false) and ($dguess \leq n/\alpha$) do:                // Iterating epochs
12         $W \leftarrow [n] \setminus \{i\}$
13         $N_a \leftarrow$ the set of neighbors of node $i$ in graph $G(dguess)$
14         $dexcess \leftarrow 1$
15         $cont \leftarrow$ true
16         while ($cont =$ true) and ($dguess \cdot dexcess \leq n/\alpha$) do:          // Iterating stages
17             $Col \leftarrow \{i\}$
18             $Close\_Col \leftarrow \{i\}$
19             for $col\_rounds \leftarrow 1$ to $10 \log_{\alpha/2}(n)$ 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                     $R \leftarrow R \cup R'$ and DELIVER($R'$)
23                     $Col \leftarrow Col \cup Col'$
24                     if $col\_rounds > 8 \log_{\alpha/2}(n)$ then $Close\_Col \leftarrow Close\_Col \cup Close\_Col'$
25                 $rcount \leftarrow rcount + 1$
26             if $dexcess \geq \alpha$ then $W \leftarrow W \setminus$ {neighbors of nodes in $Col$ in graph $G(dguess \cdot dexcess/\alpha)$}
27             if $|Close\_Col| < \frac{n}{dguess}$ then ($cont \leftarrow$ false) else      // Lines 27-31:  Working period
28                 $N_b \leftarrow$ the set of neighbors of node $i$ in graph $G(dguess \cdot dexcess)$ intersected with $W$
29                 send(RUMORS, $R$) to every process $p$ in $N_b$
30                 for every (RUMORS, $R'$) message received : DELIVER($R'$)
31                 $W \leftarrow W \setminus N_b$
32             $rcount \leftarrow rcount + 1$
33             $dexcess \leftarrow \alpha \cdot dexcess$
34         if $W = \emptyset$ then $done \leftarrow$ true
35         $dguess \leftarrow \alpha \cdot dguess$
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 $dline$.

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 $G(\mathit{dguess})$ having the same local round counter, epoch and stage number. The goal of this communication is to build two sets $\mathit{Close\_Col} \subseteq \mathit{Col}$ 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 $G(\mathit{dguess})$; 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 $G(\mathit{dguess})$ 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 $G(\mathit{dguess} \cdot \alpha)$).

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 $G(\mathit{dguess} \cdot \mathit{dexcess})$ 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 $G(\mathit{dguess} \cdot \mathit{dexcess})$ 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 $\mathit{dguess} \cdot \log^{O(1)}(n)$.

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[3]) adversarial pattern of adversary CRRI, and then we analyze its per-round message complexity. For this purpose, we fix α such that $\alpha = n^{3/\sqrt[3]{\mathit{dline}}} \geq 4$ and $\Big((10\log_{\alpha/2} n + 1) \cdot \log_\alpha n\Big) \log_\alpha n + 1 \leq \mathit{dline}$. Such value of α exists if dline is sufficiently large, for example $\mathit{dline} \geq 100$; therefore we assume it in our analysis.

In each round, we conceptually split processes into *groups* associated with a unique triple (t, g, x), corresponding to their values of $(\mathit{rcount}, \mathit{dguess}, \mathit{dexcess})$. 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.

In the analysis, of both correctness and complexity, we usually focus on processes that are

[3] Recall that for deterministic algorithms, there is no distinction between adaptive and oblivious adversary, as the adversary knows a priori how a deterministic algorithm would behave under a specific adversarial pattern.

associated with the same triple in every single round of the period considered in the analysis. We use the following methodology in different parts of the analysis: we fix a round and restrict to processes associated with some triple (t, g, x) in this round; then we proceed by analyzing some preceding round. By the update rules of *rcount* (lines 25 and 32), if processes are associated with the same triple (t, g, x) in a given round then their round counters *rcount* are also the same and decrease by one in each of $t-1$ preceding rounds. (Note that this might not be true if we considered rounds succeeding the fixed round, as some of these processes might be crashed and restarted in the future, and thus their variables *rcount* would be reset; however we don't consider succeeding rounds in our analysis.) Therefore we may skip the first coordinate of the triple from consideration, and instead we may focus on the last two coordinates, i.e., to argue in terms of *pairs* (g, x). The value t of round counter *rcount* — common for all the processes considered in the analysis — can be however useful as a reference to the round, instead of introducing round numbering according to some existing global clock (recall that although it exists, due to synchronization assumption, processes don't know it). Finally, since round counters of the analyzed processes are synchronized, so are the starting and ending points of consecutive stages, as each of them has the same length $10\log_{\alpha/2} n + 1$ and they altogether constitute a partition of the execution. Observe however that this is not the case for the starting points of epochs, since epochs may consists of different number of stages, and thus of different numbers of rounds.

6.3.1 Analysis: Correctness

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. This fact is fundamental for the correctness argument and among the crucial properties, together with the fault-tolerance of expander graphs, needed for estimating the message complexity of the algorithm.

Lemma 6.4 *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.*

Proof: Suppose, to the contrary, that p removes id of another process q during one of its epochs but q has not been sent a rumor of p since the beginning of the current epoch of process p. If the removal happened during execution of line 31 by process p, then, by the actions defined in the preceding lines 29 and 30, process p would send its rumor to process q directly. This would be a contradiction.

The only remaining case is when process p removes the id of process q from W while executing line 26. Denote by t the value of *rcount* at process p in the round when id of process q was removed from set W stored at p. Let j be the stage number of this round. Note that $j > 1$, since the condition in line 26 is satisfied only in stages bigger than 1, i.e., for *dexcess* $\geq \alpha$.

Consider the period corresponding to the current epoch of process p up to the working period of stage j of process p. We construct a sequence of processes $q_0, q_1, \ldots, q_{j^*}$, for some $1 \leq j^* \leq j-1$, satisfying the following properties:

(i) each process $q_{j'}$, for $0 \leq j' \leq j^*$, is associated with the same pair (g, x) and has the same round counter as process p in every round from the beginning of the current epoch of p till the end of stage $j - j'$ of p in the current epoch;

(ii) each process $q_{j'}$, for $0 \leq j' \leq j^*$, has received the rumor of p before its working period in stage $j - j'$ of the current epoch of process p;

(iii) each process $q_{j'}$, for $0 \leq j' \leq j^*$, removes id of process q from its set W in the working period of stage $j - j'$ of the current epoch of process p while executing line 26;

(iv) process q_{j^*} removes id of process q from its set W in the working period of stage $j - j^*$ of the current epoch of process p while executing line 31.

We start with setting $q_0 = p$. Note that process $q_0 = p$ satisfies properties (i)-(iii), as requested, though property (iv), to be satisfied by the last element of the constructed sequence, is not relevant (by definition) to process q_0. Therefore we continue construction of consecutive processes $q_{j'}$, for $1 \leq j' \leq j - 1$, until the property (iv) becomes satisfied by some node q_{j^*}.

Suppose we already constructed the sequence up to process $q_{j'}$, for some $0 \leq j' < j - 1$, and all these processes satisfy properties (i)-(iii). We show how to define process $q_{j'+1}$. By property (iii) of process $q_{j'}$ and the specification of line 26 executed by $q_{j'}$ in its stage $j - j'$ (note that it is the same stage number as in process p, by property (i)), q was a neighbor of some process in set $Col_{q_{j'},t_{j'}}$ in graph $G(dguess \cdot dexcess)$ used in the working period in the previous stage, where $t_{j'}$ is the counter of the last round of stage $j - j'$; we call this process $q_{j'+1}$.

We argue that node $q_{j'+1}$ satisfies property (i). First we show that $q_{j'+1}$ is associated with the same triple as $q_{j'}$ in this and all preceding stages of the current epoch. Indeed, it is true in stage $j - j'$ of process $q_{j'}$, by a simple inductive argument based on the condition in line 21 and the update rule of set Col in line 23. It is also true in the preceding stages of the epoch, since within the same parameter g associated with an epoch, stage parameter x changes simultaneously by the same rate in all processes that progress their stage (unless they crash, but this is not the case by stage $j - j'$ by the choice of $q_{j'}, q_{j'+1}$). The round counter in both processes $q_{j'}$ and $q_{j'+1}$ are obviously the same in all the considered rounds. Recall that, by property (i) of process $q_{j'}$, triple (t, g, x) associated with process $q_{j'}$ is the same as the corresponding triple associated with process p, in each round of this period. Hence $q_{j'+1}$ satisfies property (i).

We now argue that process $q_{j'+1}$ satisfies property (ii). More precisely, that $q_{j'+1}$ knows the rumor of p before its working period in stage $j - (j' + 1)$ of the considered epoch. Consider stage $j - j'$ of process $q_{j'}$. By the update rule in line 23 and the condition in line 21, there must be a chain of processes having the same triple (t', g, x) as $q_{j'}$ for every round counter t' within this stage such that: $q_{j'+1}, q_{j'}$ are the end-points of this chain, its length is at most $10 \log_{\alpha/2} n$, and COLLABORATE messages were propagated via the subsequent processes in the chain in the collaborating period of the considered stage. Since $j - j' > 1$ and by the fact that variable *dexcess* is updated only in line 33, which is in-between consecutive stages of the same epoch, all processes in this chain have been associated with the same pair $(g, x/\alpha)$ during every round of the previous stage (within the same epoch), and their round counters *rcount* have been synchronized in this period. By the acceptance condition in line 21, a message from $q_{j'}$ has been propagated backwards this chain to process $q_{j'+1}$ in the collaboration period of that stage (recall again that the length of this period $10 \log_{\alpha/2} n$ is not smaller than the length of the chain). Hence, during the collaborating period of this stage $j - (j' + 1)$, process $q_{j'+1}$ has received a set of rumors R that was in $q_{j'}$ in the beginning of this stage. By property (ii) of process $q_{j'}$, this set of rumors contained the rumor of p at that time. This proves property (ii) for process $q_{j'+1}$.

By the definition of process $q_{j'+1}$, it has q as a neighbor in graph $G(dguess \cdot dexcess)$ used in the working period in stage $j - (j' + 1)$ (note that these graphs are the same in both $q_{j'}$ and $q_{i'+1}$, as they are associated with the same triple in each round of this stage). There are two cases. If process $q_{j'+1}$ removed q from its set W while executing line 31 of this stage then we finish our construction and set $j^* = j' + 1$, as $q_{j'+1}$ satisfies properties (i), (ii) and (iv). Otherwise, we proceed with the inductive construction, as $q_{j'+1}$ satisfies properties (i)-(iii).

In order to complete the construction of the desired sequence, we need to show that the construction terminates after some step, i.e., that we will find a process q_{j^*}, for some $1 \leq j^* \leq j - 1$, satisfying properties (i), (ii) and (iv). Suppose, to the contrary, that we did not find such a process. It follows from the inductive step of the construction that we can construct a sequence $q_0, q_1, \ldots, q_{j-1}$ of processes satisfying properties (i)-(iii). By property (iii), process q_{j-1} removes an element, mainly q, from its set W in the working period of stage $j - (j - 1) = 1$ of process p of the current epoch, which, by property (i), is also stage 1 of q_{j-1}. This is however a contradiction, since process q_{j-1} has $dexcess = 1$ in its first stage and thus the condition allowing execution of the update in line 26 is not satisfied.

Having defined sequence $q_0, q_1, \ldots, q_{j^*}$, for some $1 \leq j^* \leq j-1$, as specified above, observe that process q_{j^*} knows the rumor of p before the working period of its stage $j - j^* \geq 1$ of the considered epoch, by property (ii). Therefore it removed q from its set W while executing line 31 in this stage. This means that process q_{j^*} had sent its known rumors, including the rumor of p, to process q directly in this period. Since this period is earlier than stage j of the current epoch of process p, we get a contradiction with the assumption from the beginning of the proof. □

We now show the correctness of algorithm det-gossip.

Theorem 6.5 *Given any adversarial pattern $\mathcal{A} \in CRRI$, algorithm* det-gossip *guarantees QoD.*

Proof: First, we note that all delivered rumors respect the deadline, since a single run of routine fixed-det-gossip lasts at most *dline* rounds: Simple calculation reveals that the length of routine fixed-det-gossip is at most $\Big((10 \log_{\alpha/2} n + 1) \cdot \log_\alpha n\Big) \log_\alpha n + 1$ rounds, since there are at most $\log_\alpha n$ epochs, each containing at most $\log_\alpha n$ stages of $10 \log_{\alpha/2} n + 1$ rounds each. Additionally there is a possibility of an extra round (lines 37-38). Since we perform this analysis for α satisfying $((10 \log_{\alpha/2} n + 1) \cdot \log_\alpha n) \log_\alpha n + 1 \leq dline$, and since $dline \leq \rho.d$, the algorithm delivers each rumor ρ by its deadline $\rho.d$.

It remains to show that each rumor is successfully delivered to all other processes that this rumor is admissible for, during a single run of the routine fixed-det-gossip. First observe that if a process executes lines 37-38 then its rumor is sent directly to all processes and thus received by all admissible ones. We prove that rumors of processes that skip lines 37-38 are also delivered to all admissible processes in the execution. More precisely, since an admissible process must be non-faulty within the deadline period, and thus also during the run of routine fixed-det-gossip for this rumor, it is enough to show that the rumor has been sent to this process in some message.

Assume that p skips lines 37-38. It follows that p finishes its execution with variable *done* set to *true*. Consequently, its set W has been emptied in the last epoch of this execution (c.f., line 34). Note that set W was re-initialized to $[n] \setminus \{p\}$ in the beginning of this epoch (line 12). By Lemma 6.4, each of these processes has been sent the rumor of p during the considered last epoch of p. This completes the proof of the theorem. □

6.3.2 Analysis: Message Complexity

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 6.8. 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 6.6 *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: Consider a round of the computation and a triple (t, g, x), for some valid integers t, g, x. Denote set $Q(t, g, x)$ by Q. There are two cases. For $\alpha \leq g \leq \alpha^2$ the lemma is obvious. Consider $g > \alpha^2$. Note that this implies that the current epoch is bigger than 2. Suppose that the statement of the lemma is incorrect, and assume that the considered round is the earliest for which it does not hold. We argue that round t must be the first round of an epoch. Otherwise we would have $Q(t, g, x) \subseteq Q(t-1, g, x/\alpha)$ or $Q(t, g, x) \subseteq Q(t-1, g, x)$, respectively, depending whether t is the first round of a stage or not (recall that starting points of stages are synchronized for processes with the same round counter), as only processes associated with the same triple with round counter $t-1$ could simultaneously upgrade their variables *dexcess* within an epoch. This however would contradict the choice of round counter t; therefore, processes in Q must all be in the beginning of their epoch corresponding to $dguess = g$ when having the round counter t.

Consider the collaborating rounds of the previous stage of processes in Q; they are alive during this period, as round t is in epoch bigger than 2. The considered period is also in the previous epoch, hence the second coordinate of the triple was g/α in each process in Q in every round of the considered period. The first coordinate — corresponding to the local round counter — is trivially the same in all processes in Q in any round of this period. Note however that some processes in Q may have pairwise different third coordinates in the considered stage. By the choice of t, we have that any subset of Q associated with the same triple in the considered period was of size at most $\log_\alpha n \cdot \alpha^2 \cdot \frac{n}{g/\alpha} = \log_\alpha n \cdot \frac{\alpha^3 \cdot n}{g}$. However, if any of these subsets, say Q', was bigger than $\alpha^2 \cdot n/g$, in the round with counter $t-1$ (i.e., in the last round of the previous epoch), we would have the following.

By Lemma 6.3 applied to graph $G(g/\alpha)$ of $(\alpha \cdot n/g)$-expansion, used in the collaborating period of the epoch preceding round t, there is a subset of at least $\alpha \cdot n/g$ processes in Q' such that the subgraph of $G(g/\alpha)$ induced by these processes has diameter at most $2\log_{\alpha/2} n$. This implies that their sets $Close\_Col_{p,t-2}$ of close collaborators at the end of the collaborating period preceding round $t-1$, containing all these processes within distance $2\log_{\alpha/2} n$, would be large enough (i.e., at least $n/(g/\alpha)$) to make these processes p stay in the previous epoch instead of increasing it at the end of round $t-1$ (see the update of the loop stopping condition in line 27). This is a contradiction with the fact that all elements in Q change their epochs at the end of round with counter $t-1$.

This proves that each subsets of Q associated with the same triple in the round with counter $t-1$ is of size at most $\alpha^2 \cdot n/g$. There are at most $\log_\alpha n$ of such disjoint subsets, as the second coordinate of a triple is the same in all processes in Q (i.e., equal to g/α), while there are at most $\log_\alpha n$ of different values on the third coordinate. Hence, $|Q| \leq \log_\alpha n \cdot \alpha^2 \cdot n/g$, which is a contradiction with the choice of the first round with counter t contradicting the lemma and the fact that $Q = Q(t, g, x)$. □

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 6.7 *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: Let Q' be the set of processes satisfying the conditions of the lemma in the definition of set U. Note that if Q' is empty then set U is also empty. Assume $Q \neq \emptyset$. Suppose, to the contrary, that set U is of size at least $\alpha \cdot n/(g \cdot x)$. Consider process p in Q'. By definition of set Q', we have $|Close\_Col_{p,t}| \geq n/g$. By expansion of graph $G(g \cdot x/\alpha)$ used in the working period of the previous stage, there is an edge between sets $Close\_Col_{p,t}$ and U in this graph. Therefore, while executing line 26 in the current stage, process p removes some element of U, call it u, from its set W.

Consider any process $q \in Q'$ different than p (if there are no such element then we would get an immediate contradiction with the definition of U, as p removed u from its set W, which in this case is equal to U). Consider set $Close\_Col_{p,t}$. By definition of Q', we have $|Close\_Col_{q,t}| \geq n/g$. Again by expansion of graph $G(g \cdot x/\alpha)$ used in the working period of the previous stage, there is an edge between sets $Close\_Col_{p,t}$ and set $Close\_Col_{q,t}$. Since both sets are still alive after the first $8 \log_{\alpha/2} n$ rounds of the coordinating period of the current stage, by specification of close neighborhoods in the pseudo code (lines 18 and 24), and both have diameter at most $4 \log_{\alpha/2} n$, we observe the following behavior during the collaborating period. Processes in set $Close\_Col_{p,t}$ exchange messages among themselves during the first at most $5 \log_{\alpha/2} n$ rounds of this period, then one of them will propagate it to some node in $Close\_Col_{q,t}$ in one round, and finally node q gets it as it gathers info from processes in $Close\_Col_{q,t}$ during the last $2 \log_{\alpha/2} n$ rounds of the considered period. All this information about processes is propagated inside variables Col. Consequently, process q removes all neighbors of processes in $Col_{q,t} \supseteq Close\_Col_{p,t}$ in graph $G(g \cdot x/\alpha)$ from its set W in line 26 before executing line 27. In particular, it removes process u, by definition of u. It follows that every process in Q' removes $u \in U$ from its set W before executing line 27 in round t, which contradicts the definition of U. Hence, U must be smaller than $\alpha \cdot n/(g \cdot x)$. □

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

Theorem 6.8 *The per-round message complexity of algorithm* det-gossip *is at most*

$$O\left(n^{1+6/\sqrt[3]{d_m}} \log^{O(1)}(n)\right),$$

d_m being the minimum deadline of any rumor active in the system.

Proof: Fix a valid triple (t, g, x). Consider a single round of the computation. It follows from Lemma 6.6 and by the specification of $\Delta(\cdot)$, that there are at most $O(\log_\alpha n \cdot \alpha^2 \cdot (n/g) \cdot \Delta(g)) =$

$O(\log_\alpha n \cdot \alpha^2 \cdot n \log^{O(1)}(n))$ COLLABORATE messages sent in this round by processes associated with the considered triple.

Recall that RUMORS messages are sent only if t corresponds to a working period or to the last round of fixed-det-gossip spent on executing lines 27-31. Now we estimate the number of RUMORS messages sent in the considered round when $x \geq \alpha$. By Lemma 6.7, the union of sets W in the processes with triple (t, g, x) that want to send RUMORS messages is at most $\alpha \cdot n/(g \cdot x)$. It follows, by the node degree of graph $G(g \cdot x)$ used for sending messages in the considered round, that each process in such union is sent at most $\Delta(g \cdot x)$ RUMORS messages in the considered round, and therefore the total number of such messages sent/received in this round is $O(\alpha \cdot n/(g \cdot x) \cdot \Delta(g \cdot x)) = O(\alpha \cdot n \log^{O(1)}(n))$.

Now we estimate the number of RUMORS messages sent in the considered round when $x = 1$. In this case, the size of set $Col_{p,t}$ must be not bigger than $\log_\alpha n \cdot \alpha^2 \cdot n/g$, by Lemma 6.6. Hence, the total number of RUMORS messages sent in round number t by processes associated with triple (t, g, x) is $O(\log_\alpha n \cdot \alpha^2 \cdot (n/g) \cdot \Delta(g \cdot x)) = O(\log_\alpha n \cdot \alpha^2 \cdot (n/g) \cdot \Delta(g \cdot 1)) = O(\log_\alpha n \cdot \alpha^2 \cdot n \log^{O(1)}(n))$.

It remains to estimate the number of RUMORS messages if t is the last round of the routine fixed-det-gossip when lines 37-38 are executed. We must have $g = n$ then, as it was upgraded from n/α at the end of the previous round (line 35). By Lemma 6.6, the number of processes associated with the same triple having n on its second coordinate is at most $\log_\alpha n \cdot \alpha^2$. The number of RUMORS messages sent by a process in lines 37-38 is n, hence the total number of such messages is at most $\log_\alpha n \cdot \alpha^2 \cdot n$.

Summarizing all three cases concerning RUMORS messages, the total number of RUMORS messages sent/received in a round is $O(\log_\alpha n \cdot \alpha^2 n \log^{O(1)}(n))$.

Finally, there are $O((\log_\alpha n)^2 \cdot \log_\alpha^3 n) = O(\log_\alpha^5 n)$ valid triples: there are at most $\log_\alpha n$ different values g of *dguess* and at most $\log_\alpha n$ different values x of *dexcess* for fixed g, while the total length of a run of the routine fixed-det-gossip is $O(\log_\alpha^3 n)$, as we argued in the beginning of the analysis section.

Summarizing, the number of messages sent (and thus also received) in one round is at most

$$\Big(O(\log_\alpha n \cdot \alpha^2 n \log^{O(1)}(n)) + O(\log_\alpha n \cdot \alpha^2 n \log^{O(1)}(n))\Big) \cdot O(\log_\alpha^5 n) = O(\alpha^2 n \log^{O(1)}(n) \cdot \log_\alpha^6 n) \ .$$

Observe that *dline* is not smaller than the upper bound on the length of the run of the routine fixed-det-gossip, which, as we showed, is $O(\log_\alpha^3 n)$, or more precisely, at most

$$\Big((10 \log_{\alpha/2} n + 1) \cdot \log_\alpha n\Big) \log_\alpha n + 1 \leq 21 \log_\alpha^3 n \ .$$

Consequently, $\log_\alpha n \leq \sqrt[3]{d_m/21}$, for $\alpha \geq 4$, and the number of messages per round is

$$O(n^{1+2/\log_\alpha n} \log^{O(1)}(n) \cdot \log_\alpha^6 n) \leq O(n^{1+6/\sqrt[3]{d_m}} \log^{O(1)}(n) \cdot d_m^2) \ .$$

If $d_m = O(\log^3 n)$ than this is bounded from above by

$$O(n^{1+6/\sqrt[3]{d_m}} \log^{O(1)}(n)) \ .$$

On the other hand, we may deliver all rumors by time $O(\log_\alpha^3 n) \leq O(\log^3 n)$, for $d_m = \Omega(\log^3 n)$, which means that the number of messages in such case can be bound from above by $O(n \log^{O(1)}(n))$. This completes the proof. □

Remark 6.9 *When the above upper bound result is contrasted with the lower bound result of Theorem 4.1 one can conclude that algorithm* det-gossip *is close to optimal. In fact, for* $d_m = \Theta(\log^{O(1)}(n))$ *algorithm* det-gossip *is optimal within log factors.*

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. Our algorithms operate under the assumption that processes do not have access to a global clock. We believe that it is possible to further improve the complexity of our algorithms (or devise new algorithms) if a global clock is assumed; then the computation could be appropriately divided into epochs of a certain size and the messages could be spread over many rounds. The global clock would help synchronize the processes over epochs. The main open challenge here is to determine the size of the epochs.

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 tradeoffs 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 [14]. 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.

References

[1] J. Aspnes: Spreading rumors rapidly despite an adversary. *J. of Algorithms*, 26: 386–411, 1998.

[2] 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.

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

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

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

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

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

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

[9] 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.

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

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

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

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

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

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

[16] 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.

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

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

[19] 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.

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

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

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

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

[24] 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.

[25] M. Saks, N. Shavit, and H. Woll. Optimal time randomized consensus–making resilient algorithms fast in practice. In *SODA 1991,* pages 351–362.

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

Appendix

Proof of Fact 5.1. Choose some node $s \in V'$, and fix s for the remainder of the proof. Let $N_s(G, i)$ be all the nodes in V' within distance i of s in graph G. We argue that $N_s(G, \log_\gamma k) = k$, which implies that the diameter is at most $\log_\gamma k$.

For each $v \in V'$, we (arbitrarily) divide the edges selected by v into two sets $E_A(v)$ and $E_B(v)$, each of size $8c(n/k)\gamma \log^2 n$. Let E_A be the union of edges $E_A(v)$ for every $v \in V'$; let E_B be the union of edges $E_B(v)$ for every $v \in V'$. Let $G_A = (V', E_A)$, and let $G_B = (V', E_B)$.

We first examine graph G_A and argue that $N_s(G_A, \log_\gamma k - 1) \geq k/4$, i.e., there are at least $k/4$ nodes within distance $\log_\gamma k - 1$ of s.

Assume, for the sake of contradiction, that $N_s(G_A, \log_\gamma k - 1) < k/4$. Let B_i be the set of nodes that are in $N_s(G_A, i)$ but are not in $N_s(G_A, i-1)$. That is, $B_i = \{v \in N_s(G_A, i) \setminus N_s(G_A, i-1)\}$. Define $B_0 = \{s\}$. Note that $B_0 \cup B_1 \cup \ldots \cup B_i = N_s(G_A, i)$. Notice that the assumption that $N_s(G_A, \log_\gamma k - 1) < k/4$ also implies that $|B_{\log_\gamma k - 1}| < k/4$.

We now prove the following invariant: $|B_i| \geq \gamma^i$ for all $i < \log_\gamma k$; this leads to a contradiction, as it implies that $B_{\log_\gamma k - 1} \geq k/2$. The base case for B_0 is clear; we proceed by induction. Assume, for the sake of induction, that $|B_i| \geq \gamma^i$.

We now examine the outgoing edges from the nodes in B_i in E_A; observe that there are at least $|B_i| \cdot 8c(n/k)\gamma \log^2 n \geq 8c(n/k)\gamma^{i+1} \log^2 n$ such edges. If these edges do not induce a set B_{i+1} containing more than γ^{i+1}, then there must be some set S of size γ^{i+1} such that every one of these edges leads to a node in S or $N_s(G_A, i)$ or $V \setminus V'$. The probability of this event can be bounded as:

$$\begin{aligned}
&\leq \quad (\#\text{ of possible sets } S \text{ of size } \gamma^{i+1}) \cdot \Pr(\text{one edge hits } S \text{ or } N_s(G_A, i) \text{ or } V \setminus V')^{8c(n/k)\gamma^{i+1}\log^2 n} \\
&\leq \quad \binom{k - N_s(G_A, i)}{\gamma^{i+1}} \cdot \left(1 - \frac{k/2}{n}\right)^{8c(n/k)\gamma^{i+1}\log^2 n} \leq k^{\gamma^{i+1}} 2^{-4c\gamma^{i+1}\log^2 n} \leq 1/n^{4c}.
\end{aligned}$$

Thus, with high probability, $|B_{i+1}| > \gamma^{i+1}$. Taking a union bound over each of the $\log_\gamma k$ steps, we conclude that this holds for all $i < \log_d k$ with probability at least $1 - 1/n^{2c}$. Since this is a contradiction, we conclude that $N_s(G_A, \log_\gamma k) > k/4$ with high probability.

We now argue that every node $v \in V' \setminus N_s(G_A, \log_\gamma k)$ has an edge in E_B connecting it to some $w \in N_s(G_A, \log_\gamma k - 1)$. Fix some $v \in V' \setminus N_s(G_A, \log_\gamma k - 1)$. Note that since there are at least $k/4$ nodes in $N_s(G_A, \log_\gamma k)$, there are $2cn\gamma \log^2 n$ edges to examine. Each edge hits v with probability $1/n$, and hence the probability that v does not have a neighbor in $N_s(G_A, \log_\gamma k - 1)$ is bounded as:

$$\leq \quad (1 - 1/n)^{2cn\gamma \log^2 n} \leq 1/n^{2c}.$$

Thus, with probability $1 - 1/n^c$, graph $G = (V', E_A \cup E_B)$ has diameter at most $\log_\gamma k$, as every node in V' is within distance $\log_\gamma k$ of s. □