Simple and Optimal Fault-Tolerant Rumor Spreading

Benjamin Doerr[1], 2, Carola Doerr[1,3] Shay Moran[1], Shlomo Moran[4]

[1]Max Planck Institute for Informatics, Saarbrücken, Germany
[2]LIX, École Polytechnique, Palaiseau, France
[3]Université Paris Diderot - Paris 7, LIAFA, Paris, France
[4]Computer Science Dept., Technion - Israel Institute of Technology, Haifa, 32000 Israel.
Part of this work was done while visiting Max Planck Institute for Informatics

January 8, 2014

Abstract

We present rumor spreading protocols for the complete graph topology that are robust against an arbitrary number of adversarial initial node failures. Our protocols are the first rumor spreading protocols combining the following three properties: they can tolerate any number of failures, they distribute the rumor to all nodes using linear number of messages (actually they use strictly minimal $n-1$ messages), and if an arbitrarily small constant fraction p of nodes (including the initiator of the rumor) are working correctly, our protocols communicate the rumor to all members in the network in $O(\log(n))$ rounds.

Our protocols are simpler than previous fault-tolerant rumor spreading protocols in this model, they do not require synchronization (i.e., their correctness is independent on the relative speeds of the nodes), and they do not require a simultaneous wakeup of all nodes at time 0.

Keywords: Rumor Spreading; Randomized Algorithms; Robustness; Analysis of Algorithms.

1 Introduction

Disseminating information to all nodes of a network is one of the basic communication primitives. Basically all collaborative actions in networks imply that some information has to be sent to all nodes, and surprisingly complex tasks like computing aggregates can be reduced to essentially solving a dissemination problem [MAS06]. We are interested in disseminating a single piece of information (the *rumor*) to all n nodes in a communication network in which all nodes can exchange information with each other but where individual nodes can initially crash (i.e., they do not participate in the rumor spreading process). More precisely, we study dissemination protocols that are robust against *adversarial initial node crashes* [TKM89].

1.1 Previous Results

Rumor spreading protocols that are robust against adversarial node failures have been studied, to the best of our knowledge, mainly in complete communication networks. In such networks, essentially two types of fault-tolerant rumor spreading protocols have been proposed: (i) whispering protocols, which assume a stricter communication model and achieve smaller message complexity, but require rather intricate constructions, and (ii) gossip-based protocols which build on the paradigm that nodes call randomly chosen others. The latter, due to their randomized nature, usually are highly robust against all kinds of faults, typically at the price of a higher communication effort and non-trivial termination criteria. Usually, both types of these fault-tolerant protocols require the network to be synchronized.

1.1.1 Whispering Protocols

It is easy to see that there are fault-free protocols disseminating a rumor in $\lceil \log_2(n) \rceil$ communication rounds using a total of $n-1$ messages and that both these measures are strictly optimal. A simple protocol for $n = 2^k$ nodes indexed by the numbers from 0 to $n-1$ would be that in round i, each node x having the rumor calls node x XOR 2^{i-1} and forwards the message to it. From the sender ID the recipient of a message can infer the round number i, and thus decide when to stop forwarding the message. Hence this protocol indeed uses only $n-1$ messages in total. This algorithm (like the other ones mentioned in this subsection, but in contrast to gossip-based algorithms discussed further below) maintains the *whispering property* [GP96, DP00]: in each round, the edges along which the rumor is transferred form a matching. It has two further advantages, namely (i) it requires no synchronization, in the sense that its correctness does not assume the existence of a global clock, or any restriction on the relative speeds of the processors, and (ii) nodes know when to stop forwarding the rumor.

The downside of this simple approach is that it is not at all robust. If a node is not available ("crashed"), then all other nodes that would be been informed via it will remain uninformed. This problem was overcome in the preliminary version of [GP96], which presents a protocol that is strictly optimal if no failures occur; however, when arbitrary f nodes do not participate in the collaborative process, then—instead of nodes remaining uninformed—only the runtime increases to at most $f + \lceil \log_2(n-f) \rceil$. The number of messages sent in all cases is $n-1$. In this result, as in other fault-tolerant rumor spreading algorithms, it is assumed that a node calling a crashed node learns that his call was unsuccessful.

The increase of the runtime bound (equal to basically the number of crashed nodes) of the above protocol is dissatisfactory. Subsequent fault-tolerant whispering protocols reduced this running time by adding an *opening phase*[1] which precedes the actual spreading of the rumor, and connects a large portion of the non-faulty processors in an appropriate subnetwork. The added opening phase comes with a price: (i) it is tailored for a linear lower bound αn on the number of non-faulty processors, which needs to be determined by the user in advance; hence, when the number of non-faulty processors is smaller than αn, these protocols may run into a deadlock and the rumor is not guaranteed to reach all non-faulty processors; also, the actual running time and number of messages of these protocols are determined by the αn bound rather than by the actual number of faulty processors; (ii) unlike the fault-free whispering protocols, it does assume a global clock and requires synchronization; (iii) it requires that all the non-faulty nodes are simultaneously activated at time 0.

[GP96] introduces such an opening phase which runs in $O(\log^2(n))$ time. A more intricate opening phase was later introduced in [DP00]; to the best of our knowledge, [DP00] is the only published paper which shows that for any fixed constant ε there is a protocol which can tolerate up to εn node failures, and whose time and message complexities are both asymptotically optimal. The opening phase of [DP00] uses an intricate construction, which requires that a certain, rather complex, virtual expander is stored by the network nodes during the system setup in a preprocessing phase. This construction is based on the explicit expanders of [LPS88], and on the properties of these expanders presented in [Upf94].

[1]We distinguish between *opening phase*, which is repeated each time a rumor is spread, and a *preprocessing phase*, which is performed only once, when the network is established.

1.1.2 Gossip-based Protocols

Gossip-based communication protocols build on the paradigm that nodes of a network call random neighbors and communicate with them. This is also called *randomized rumor spreading*. Randomized rumor spreading has been analyzed in various variants for different network topologies. Despite the very simple approach of talking to random neighbors, these protocols often achieve a surprisingly good runtime combined with extreme robustness. Their main advantage over the fault-tolerant whispering protocol of [DP00] is that they avoid the need for an opening phase and for storing intricate subnetworks in a preprocessing time, but, on the negative side, they do require asymptotically larger message complexities, and they typically lack a simple termination criterion (i.e., the nodes do not know when every node is guaranteed to have learned the rumor so that they can stop spreading the message). In this section, we briefly describe the results that are relevant for our work on robustness against adversarial failures in complete graphs.

The first rumor spreading result is due to Frieze and Grimmett [FG85], who studied the simple protocol consisting of each informed node in each round calling a random neighbor (*synchronized push-protocol*). Pittel [Pit87] showed that the round complexity of this protocol is $\log_2(n) + \ln(n) + h(n)$, where $h(n)$ is any function tending to infinity. Note that randomized rumor spreading in this push-model violates the whispering property, but when counting only messages which carry the rumor (see Section 2.1), this violation can be undone by assuming that nodes accept only one incoming call.

The first to analyze rumor spreading as communication protocol (namely in the context of maintaining the consistency of replicated databases) were Demers et al. [DGH$^+$88]. In applications like this, where one may assume that updates are to be disseminated frequently, also a push-pull randomized rumor spreading protocol makes sense. Here all nodes (and not only those already knowing the rumor) call random neighbors, allowing that uninformed nodes "pull" information from informed ones. Naturally, for the use of pull operations not to generate an enormous message overhead, one needs the assumption that sufficiently often rumors are injected.

A possible weakness of protocols using randomized pull operations is the inherent violation of the whispering property: Many uninformed nodes may randomly select the same informed one, thus forcing the selected node to forward the rumor to many neighboring nodes in a single round.[2] In models which allow a processor to send at most one message each round, this may result in a considerable increase in running time.

While always randomized rumor spreading is described as highly robust, not too many proofs of this statement exist. Elsässer and Sauerwald in [ES09] prove for general graphs that messages failing independently with probability $c < 1$ lead to an increase of the runtime of at most $O(1/(1-c))$. In [DHL09], this was made more precise for complete network topologies by showing that the push variant of randomized rumor spreading, with high probability, forwards a rumor to all nodes in time $\log_{2-c}(n) + (\ln n)/(1-c) + o(\log n)$, when each call independently fails with probability c. It is not difficult to see that the same bound (for the time needed to inform the working nodes) holds in complete networks also for an adversarial failure model in which a fraction of $f = cn$ of arbitrary nodes is crashed initially.

As mentioned above, the robustness of randomized rumor spreading comes at a price not only of a slightly higher dissemination time when no failures occur, but more importantly at a relatively large number of messages sent until the rumor is disseminated, and at a large

[2]Using, e.g., [RS98], it is not hard to see that in a push-pull gossip-based algorithm, in each round in which the fraction of informed processors is bounded away from 0 and from 1, w.h.p. some informed processor sends the rumor to $\Omega(\frac{\log n}{\log\log n})$ uninformed neighbors.

number of additional messages caused by the fact that in the basic protocol the nodes do not know when to stop sending out messages: In independent randomized rumor spreading in the push-model, only after $\Theta(n \log n)$ messages are sent, the rumor is known to all vertices. By adding suitable dependencies to the random choice of the communication partner, in [DF11a] a randomized rumor spreading protocol was designed which in $(1 + o(1)) \log_2 n$ rounds and with $nh(n)$ messages solves the dissemination problem (here again h is an arbitrary function tending to infinity). This protocol is robust against random node crashes (leading to a time bound of $(1+o(1)) \log_{2-c} n$ if $f = cn$ nodes are crashed). The protocol has a simple termination criterion, that is, the bounds on the number of messages not only refer to the messages sent until all nodes are informed, but in fact to the total number of messages sent in one run of the protocol. The protocol can be made robust also against adversarial node crashes, however, at the price of the message complexity increasing to $\Theta(n \log n)$ when a constant fraction of adversarial node crashes has to be tolerated.

1.2 Our Results

In this work we present simple and efficient fault-tolerant whispering protocols, which instead of using expanders add a natural randomization to the elegant (but sub-optimal) whispering protocol of [GP96]. The resulted protocols:

- Are robust against any number of failures.
- Do not need to construct and store, in a preprocessing time, an intricate network structure.
- Do not need an opening phase, or simultaneous wakeup of all processors at time 0.
- Use only push operations.
- Are asynchronous, in the sense that they do not need a global clock or synchronization.
- Have a very simple termination criterion.
- Maintain always strictly optimal message complexity (i.e., use $n - 1$ messages to inform all non-crashed nodes in the network) and achieve w.h.p. an asymptotically optimal runtime.[3] The runtime and message complexities are determined by the actual number of the non-faulty processors in a run, and not by a predetermined lower bound on this number.

We first show that for random node crashes, the basic protocol of [GP96], denoted GP, has a much better performance than what the worst-case bound in [GP96] states. In particular, when each node is crashed with constant probability $0 < 1 - p < 1$ (independently at random), then with high probability the algorithm terminates within $\Theta((\log(n))$ rounds. As a straightforward probabilistic analysis of the protocol of [GP96] appears tricky, we prove this result by first introducing an intermediate failure model, the wakeup model, and then coupling the two models by embedding them in the standard σ-algebra of infinite binary sequences. We believe that the wakeup model itself is of independent interest.

For adversarial node failures, this implies the following *randomized* solution: The start node (i.e., the node which the rumor starts at) picks a random permutation of the other nodes and initiates the GP protocol with node labels permuted according to this permutation. This gives the same time bounds as for random node failures. The downside is that to make the other

[3]We did not try hard to optimize the constants in the runtime bounds, but a bound of $(6 + o(1)) \log_2 n/p$, where pn is the number of non-faulty processors, follows easily from our proofs.

nodes adopt this strategy, sufficient information on the permutation chosen by the initial node has to be communicated to the other nodes as well. This can be achieved by adding a total of $O(n \log^2 n)$ bits to all the messages, with at most n bits to some messages. The total overhead is comparable to the one of fault-tolerant gossip based protocols (which use $\Theta(n \log(n))$ messages of $\Theta(\log(n))$ bits each), but is asymptotically larger than the overhead of the whispering protocol of [DP00].

When communication is costly, message sizes can be shortened to $O(\log n)$ (which is the messages size in the protocol of [DP00]): We prove that instead of choosing the permutation randomly from all permutations, it suffices to choose the permutation randomly from a set of only $O(nh(n)/\log n)$ *random* permutations, where $h \in \omega(1)$ is an arbitrary function tending to infinity. (The number of permutations can be varied to adjust runtimes and failure probabilities, see Theorem 5.2 for the details.) This allows to encode the permutation via only $\Theta(\log n)$ bits (which is also the overhead of the original GP protocol). This approach can be implemented by computing, for an arbitrary function $h \in \omega(1)$, $O(nh(n)/\log n)$ random permutations and storing them at all processors, and repeating this procedure whenever processors join or leave the network. Thus, this protocol is particularly appealing when communication is expensive, memory is cheap, and processors are not added or removed from the network too often. We conclude by noting that this preprocessing stage may be eliminated if an appropriate set of permutations can be found efficiently by a deterministic method.

2 Preliminaries

Before we present a few basics about rumor spreading protocols, let us briefly fix the notation used throughout this work. Unless stated otherwise, we consider executions of rumor spreading algorithms by n processors ordered by their names $(0, 1, \ldots, n-1)$, where 0 is the start processor.

We use the following notation: For a sequence $s = (s_1, s_2, \ldots)$, $\mathbf{odd}(s) = (s_1, s_3, \ldots)$ is the subsequence of the odd indexed elements of s, and $\mathbf{even}(s) = (s_2, s_4, \ldots)$ is the subsequence of the even-indexed elements of s. For a binary vector $\vec{b}$, $|\vec{b}|_0$ is the number of zeros in $\vec{b}$, and $|\vec{b}|_1$ denotes the number of ones in $\vec{b}$.

For a rooted tree T, $\text{height}(T)$ is the height of T, i.e., the maximum length of a path from the root to a leaf.

For $n \in \mathbb{N}$ ($\mathbb{N}$ denotes the positive integers) we abbreviate $[n] := \{1, 2, \ldots, n\}$. By S_n we denote the set of all permutations of the set $[n]$.

By ln we denote the natural logarithm to base e. All other logarithms are to base 2.

2.1 Rumor Spreading Protocols

To ease the comparison of our rumor spreading protocol with previous ones, let us briefly give a unified description of these. Let the undirected graph $G = (V, E)$ describe the underlying communication network, that is, nodes of this graph represent processors and a direct communication between two processors is possible if and only if there is an edge between the corresponding nodes. Let $n := |V|$.

A (synchronous) execution of a rumor-spreading algorithm on G consists of rounds $\mathbb{R}_1, \mathbb{R}_2, \ldots$. A round $\mathbb{R}_t$ is initiated by a set of processors $V_t \subseteq V$ (the exact nature of V_t depends on the model assumed and/or on the specific algorithm): each processor $u \in V_t$ sends a (u, v) *communication request* (in short "(u, v) request") to one of its neighbors v; the request contains a bit informing v whether u holds the rumor already. A (u, v) request is *valid* if exactly one of u and v holds the rumor. After receiving all the requests sent to it at $\mathbb{R}_t$, each processor v may (but does not have to) *approve* some of the valid requests that it has received. The round

$\mathbb{R}_t$ is then completed by transferring the rumor along the edges of the approved requests.[4] The execution *terminates* at time t if $V_t \neq \emptyset$ and $V_{t+1} = \emptyset$. We call t the *time* (or *round*) *complexity* of the execution of the rumor spreading algorithm. Note that in some other works, in particular those on gossip based randomized rumor spreading, only the first time at which all processors know the rumor is regarded. A rumor spreading protocol is *asynchronous* if the processors local programs do not use round numbers (or on any other notion of time). All the protocols proposed in this work are asynchronous.

Let E_t denote the set of edges along which requests are sent in $\mathbb{R}_t$, and let F_t denote the set of edges of the approved requests, along which the rumor is transferred at $\mathbb{R}_t$ (thus $|E_t| = |V_t| \leq n$ and $F_t \subseteq E_t$). A rumor spreading algorithm satisfies the *whispering property* if F_t always forms a matching, meaning that each processor may either send or receive at most one copy of the rumor at each round. Some authors actually require any rumor spreading algorithm to satisfy the whispering property (see, e.g., [GP96, DP00]).

Besides the time complexity, the communication effort and the robustness against faults are two further important performance measures. There are some variants of the definition of *message complexity* of rumor spreading algorithms. The strictest definition counts all communication requests, i.e. $\sum_t |E_t|$ (e.g. [GP96, DF11b]). A more permissive definition assumes that communication between uninformed processors is given for free due to frequent injections of other rumors ([KSSV00, CHHKM12]), and hence it reduces to $\sum_t |\{(u,v) : (u,v) \in E_t \text{ and either } u \text{ or } v \text{ holds the rumor }\}|$. As will be noted soon, our algorithms have the minimum possible message complexity by both definitions.

The *faults* assumed in this paper are *initial crash failures*: A processor is faulty in a given execution if it never sends a message during the execution. We consider two types of failure policies, associated with a success parameter $p \in (0,1)$: *random failures*, in which each process may fail independently with probability $1-p$, and *adversarial failures*, in which the adversary may fail (before the execution of the algorithm starts) any subset of up to $(1-p)n$ processors, excluding the start processor. An (i,j) request is *failed* if j is faulty, and it is *successful* otherwise. Note that in our synchronized model, a faulty node j is identified by not responding to an (i,j) request.

2.2 The Algorithm of Gasieniec and Pelc

We use the following variant of the divide-and-conquer algorithm of Gasieniec and Pelc [GP96], to be denoted GP. Initially the start processor 0 holds a list $(1, 2, \ldots, n-1)$ of all uninformed processors, and all other processors hold empty lists. At each round, each processor i which holds a nonempty list $(j_1, \ldots, j_k)$, sends an (i, j_1) request and deletes j_1 from its list. If the request is successful then i also sends to j_1 the rumor, appends to it the list $\mathbf{even}(j_2, \ldots, j_k) = (j_3, j_5, \ldots)$, and sets its own list to $\mathbf{odd}(j_2, \ldots, j_k)$. Thus, in this case, the next round starts with i holding the list $\mathbf{odd}(j_2, \ldots, j_k)$ and processor j_1 holding the list $\mathbf{even}(j_2, \ldots, j_k)$. The algorithm terminates when all processors hold empty lists.
Implementation note: Observe that each list of the form $\mathbf{even}(j_2, \ldots, j_k)$ generated during the algorithm is an arithmetic progression whose difference is 2^m for some integer $m \leq \log n$. Sending such a list can be done by sending the first element j_3, the length of the list $\lfloor \frac{k-1}{2} \rfloor$, and the exponent m. Overall, this requires an addition of less than $3 \log n$ bits to the rumor.

Note that this protocol automatically ensures that (i) each node receives at most one communication request per round (hence the whispering property is satisfied), (ii) only requests from informed nodes to uninformed ones are issued (hence there is no reason not to approve a

[4]Some models assume that an informed processor u always sends the rumor on the selected edge (u,v), even if v is already informed.

request), and (iii) the protocol terminates as soon as all processors know the rumor.

The optimality of the message complexity of the GP algorithm (under the different variants of "message complexity" discussed in Section 2.1) is implied by the following straightforward observation.

Lemma 2.1 ([GP96]). *The* GP *algorithm performs the minimum possible number of communication requests, namely $n-1$ communication requests in each possible execution.*

In the presence of f crashed nodes, the time complexity of the GP algorithm is given by the following lemma.

Lemma 2.2 ([GP96]). *For up to f initial node failures the time complexity of the* GP *algorithm is at most $f + \lceil \log(n-f) \rceil$. This bound is tight if processors $1, \dots, f$ are failed.*

2.3 Reminder: Chernoff's Bounds

We apply several versions of Chernoff's bound. The following can be found, for example, in [DP09].

Theorem 2.3 (Chernoff's bounds). *Let $X = \sum_{i=1}^{n} X_i$ be the sum of n independently distributed random variables X_i, where each variable X_i takes values in $[0,1]$. Then the following statements hold.*

$$\forall t > 0 : \Pr[X > \mathrm{E}[X] + t] \leq \exp(-2t^2/n) \textit{ and} \tag{1}$$
$$\Pr[X < \mathrm{E}[X] - t] \leq \exp(-2t^2/n)\,.$$
$$\forall \varepsilon > 0 : \Pr\left[X < (1-\varepsilon)\,\mathrm{E}[X]\right] \leq \exp\left(-\varepsilon^2\,\mathrm{E}[X]/2\right) \textit{ and} \tag{2}$$
$$\Pr\left[X > (1+\varepsilon)\,\mathrm{E}[X]\right] \leq \exp\left(-\varepsilon^2\,\mathrm{E}[X]/3\right)\,.$$
$$\forall t > 2e\,\mathrm{E}[X] : \Pr[X > t] \leq 2^{-t}\,. \tag{3}$$

Chernoff's bound applies also to random geometric variables. A proof of the following theorem can be found, e.g., in [AD11, Theorem 1.14].

Theorem 2.4 (Chernoff's bound for random geometric variables). *Let $p \in (0,1)$. Let $X_1, \dots, X_n$ be independent geometric random variables with $\Pr[X_i = k] = (1-p)^{k-1}p$ for all $k \in \mathbb{N}$. Let $X := \sum_{i=1}^{n} X_i$.*

Then for all $\delta > 0$, $\Pr[X \geq (1+\delta)\,\mathrm{E}[X]] \leq \exp\left(-\frac{\delta^2(n-1)}{2(1+\delta)}\right)$.

3 Random Failure Analysis of the GP Algorithm via a New Random Wakeup Model

In this section we show that the GP algorithm has a much better performance against *random* node failures than the worst case performance given in Lemma 2.2 against *adversarial* node failures. We assume that each processor may fail with probability $1-p$ independently. It is not hard to see that the expected runtime is bounded from below by the solution to the recursive formula $F(1) = 0; F(n) = p \cdot F(n/2) + (1-p) \cdot F(n-1) + 1$, which is $\log(n)/p + O(1)$. On the other hand, we show that every processor is informed after $3.5 \log(n)/p$ rounds, with high probability.

We refer to p as the *success rate,* and to $1-p$ as the *failure rate.* Due to the sequential nature of the GP protocol, even a very small change in the failure pattern (that is, the set of failed nodes) may imply a large change in the time complexity. This makes a straightforward analysis of this model a bit tricky. To ease the analysis, we start by considering a similar protocol in a simpler model, the *random wakeup model*, which we believe to be of independent interest. We then transfer the results to the standard random node failure model by coupling the models in the standard σ-algebra of infinite binary sequences.

3.1 The Random Wakeup Model

We regard the following divide-and-conquer wake-up protocol, which is inspired by the GP algorithm. The start processor 0 starts with the list $(1, 2, \ldots, n-1)$ of nodes to be informed. It sends in every round a communication ("wakeup") requests to processor 1, until this processor is woken up. It then forwards to it the rumor, appended by the list $\mathbf{even}(2, ..., n-1)$, thus keeping for itself the list $\mathbf{odd}(2, ..., n-1)$ as its todo-list. It then tries to wake up processor 2 in the next round, and so on. In this model, each wakeup request is successful with probability p, independently of previous requests. Hence in the implied rumor spreading algorithm, to be denoted WU, whenever u selects an edge (u, v), it repeatedly sends (u, v) requests until v is woken up. Informally, the time complexity of the algorithm in this model is larger than in the standard initial-failures model, since in the standard model only one request is sent to each processor. A formal proof of this statement is given in Section 3.3.2. Note also that like the GP algorithm, the WU algorithm performs the minimum possible number of communication requests in each execution: $n + f - 1$ requests when there are f failed wakeup messages.

The time complexity of the random wakeup model is easier to analyze since the implied WU algorithm sends communication requests along a fixed set of edges, which is independent of the specific failure pattern. For analyzing this time complexity we represent the WU algorithm by a full binary tree $\mathcal{T}$ with n leaves, in which each vertex x is labeled by a processor name $\mathsf{L}(x) \in \{0, \ldots, n-1\}$ according to the following scheme (cf. Figure 1). The leaves of $\mathcal{T}$ are labeled by the processor names $0, \ldots, n-1$, according to some arbitrary but fixed order. The labeling of an internal vertex x with children y, z is $\mathsf{L}(x) = \min\{\mathsf{L}(y), \mathsf{L}(z)\}$. Thus $\mathsf{L}(r) = 0$ (where r is the root of the tree), and for each processor k, the vertices of $\mathcal{T}$ labeled by k form a directed path, Path_k, ending at a leaf of $\mathcal{T}$.

The algorithm for processor $k \in \{0, \ldots, n-1\}$ implied by the above labeled tree $\mathcal{T}$ is the following: After receiving the rumor, k moves along the vertices of Path_k. When k steps on a non-leaf vertex $x \in \mathrm{Path}_k$ with children y, z, it repeatedly sends communication requests to $j = \max\{\mathsf{L}(y), \mathsf{L}(z)\}$ until j wakes up.

Consider now a specific execution $\mathcal{E}_{\mathrm{WU}}$ of the above random wakeup algorithm. For each internal vertex $x \in \mathcal{T}$ with children y, z, let $k_x = \mathsf{L}(x)$ and $j_x = \max\{\mathsf{L}(y), \mathsf{L}(z)\}$. Denote by $\mathrm{Delay}(x)$ the number of (k_x, j_x) requests sent by k_x in $\mathcal{E}_{\mathrm{WU}}$. Then $\mathrm{Delay}(x)$ is a geometric random variable with probability p, that is $\Pr[\mathrm{Delay}(x) = \ell] = (1-p)^{\ell-1}p$ for all positive integers ℓ, and $\mathrm{E}(\mathrm{Delay}(x)) = 1/p$.

For a processor $j \in [0 \ldots n-1]$, let P_j be the path from the root r of $\mathcal{T}$ to the (unique) leaf labeled by j, and let $\mathrm{Delay}(P_j) := \sum_{x \in P_j} \mathrm{Delay}(x)$. Then the time complexity of $\mathcal{E}_{\mathrm{WU}}$ is given by

$$\mathrm{time}(\mathcal{E}_{\mathrm{WU}}) = \max_{j \in [0 \ldots n-1]} \{\mathrm{Delay}(P_j)\}.$$

3.2 The Time Complexity of the Random Wakeup Model

Theorem 3.1. *Let $c > 1$ be a constant and let $p \in (0, 1)$ be arbitrary (possibly $p = 1 - o(1)$).*

With probability at least $1 - n \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right)$, the WU *algorithm with success rate p has delivered the rumor to all processors after $\frac{c}{p}(\lceil \log(n-1) \rceil + 1)$ rounds.*

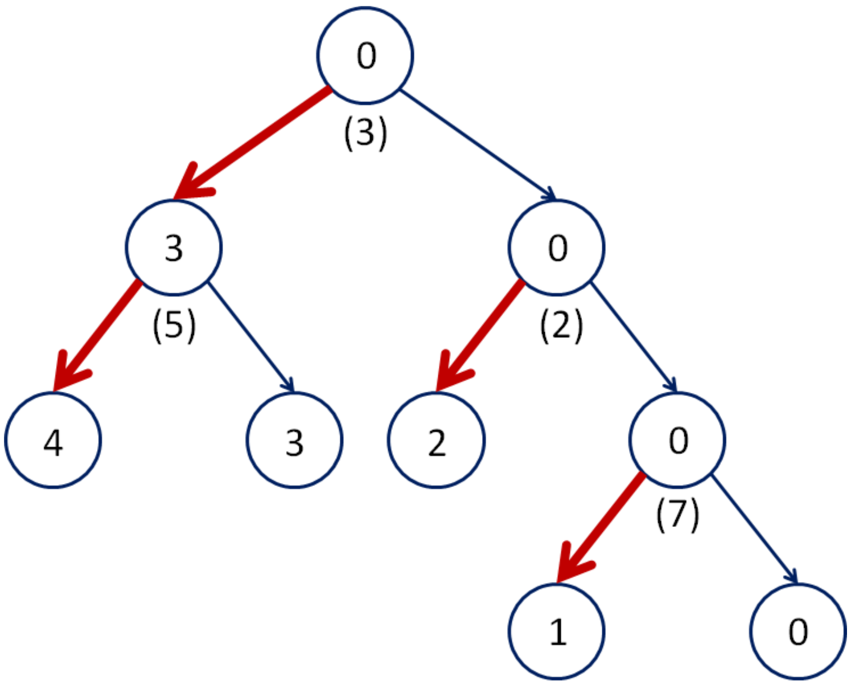

Figure 1: **Illustration of the rumor spreading in the random wakeup model for 5 processors:** Each vertex of $\mathcal{T}$ is labeled by a processors name. The red bold edges indicate rumor transfers. Thus Processor 0 always transfers the rumor to processors 3, 2, and 1 (in this order). The numbers in parentheses beneath internal vertices indicate the number of wakeup calls in a specific execution. That is, in the depicted execution processor 1 woke up only by the $7th$ $(0,1)$ request. The time complexity of this execution is $3+2+7=12$.

The success probability in Theorem 3.1 becomes $1-o(1)$ for c with $\frac{(c-1)^2}{2c\ln 2} > 1$, e.g., for $c \geq 7/2$. The theorem follows essentially from the Chernoff bound for random geometric variables, cf. Theorem 2.4.

Proof of Theorem 3.1. By construction, for each processor $j \in [0\dots n-1]$ we have that path P_j has at least $\lceil \log(n-1) \rceil$ and at most $\lceil \log(n-1) \rceil + 1$ nodes. Therefore the expected delay of path P_j, $\mathrm{E}[\mathrm{Delay}(P_j)]$, equals $(1/p)\lceil \log(n-1) \rceil$ or $(1/p)(\lceil \log(n-1) \rceil + 1)$, respectively. Since the variables $\{\mathrm{Delay}(x) : x \in P_j\}$ are mutually independent, by Theorem 2.4 we have

$$\begin{aligned}\Pr[\mathrm{Delay}(P_j) > (c/p)(\lceil \log(n-1) \rceil + 1)] &\leq \Pr[\mathrm{Delay}(P_j) > (1+(c-1))\,\mathrm{E}[\mathrm{Delay}(P_j)]] \\ &\leq \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right).\end{aligned}$$

A simple union bound over all n paths concludes the proof. □

3.3 Coupling the GP and WU Models

To relate the time complexities of the random wakeup model and the GP algorithm in the presence of random node failures, we embed the failure patterns of both models in the probability space consisting of infinite binary vectors $\{\vec{b} \mid \vec{b} \in \{0,1\}^{\mathbb{N}}\}$, where the entries $\vec{b}_1, \vec{b}_2, \dots$ are i.i.d. with a Bernoulli distribution parametrized by the success rate p—see, e.g., Chapter 2 of [Bil95]. (Infinite sequences are needed since the number of possible failures in executions of the WU algorithm is unbounded). This embedding of failure patterns induces distributions over executions of rumor spreading algorithms, similarly to the way randomized algorithms are presented in the classical work of Yao [Yao77].

In Section 3.3.1 we define the mappings of infinite binary vectors to failure patterns, and then to execution trees (whose heights represent time complexity of the corresponding execution of the GP and the WU protocol, respectively). In Section 3.3.2 we use this mapping to present our coupling argument (Lemma 3.3).

Section 3.3.1 is quite technical. The reader only interested in the main results may want to skip these details and jump directly to Section 3.3.2, considering $\text{height}(T_{\text{GP}}(n, \vec{b}))$ and $\text{height}(T_{\text{WU}}(n, \vec{b}))$ defined to be the time complexity of one particular execution of the GP algorithm and the WU algorithm, respectively.

3.3.1 Failure Patterns and Execution Trees

Any execution of the GP or of the WU algorithm with a single start processor is determined by the initial *system configuration* (in short *configuration*). A configuration is a pair $(n, \vec{b})$, where n is the number of processors to which the rumor has to be delivered, and $\vec{b} = (b_1, b_2, \ldots)$ is an infinite binary vector representing a *failure pattern.* An entry $b_i = 0$ corresponds to a failed request and $b_i = 1$ corresponds to a successful request. For each configuration $(n, \vec{b})$, $\mathcal{E}_{\text{GP}}(n, \vec{b})$ denotes the execution of the GP algorithm on $(n, \vec{b})$, and $\mathcal{E}_{\text{WU}}(n, \vec{b})$ denotes the execution of the WU algorithm on $(n, \vec{b})$ ($\mathcal{E}_{\text{GP}}(n, \vec{b})$ is always determined by the first n bits of $\vec{b}$, while $\mathcal{E}_{\text{WU}}(n, \vec{b})$ is usually determined by a longer prefix of $\vec{b}$).

$\mathcal{E}_{\text{GP}}(n, \vec{b})$ is defined by the *execution tree* $T_{\text{GP}}(n, \vec{b})$ as follows. The vertices of $T_{\text{GP}}(n, \vec{b})$ are configurations. The root of $T_{\text{GP}}(n, \vec{b})$ is the configuration $(n, \vec{b})$. If $\vec{b} = 0\vec{c}$ (for some infinite binary vector $\vec{c}$) then the first request sent by the execution failed. Hence in the next round there is still only one informed processor, with $n-1$ uninformed processors in its list. Thus the only child of $(n, \vec{b})$ is $(n-1, \vec{c})$. If $\vec{b} = 1\vec{c}$ then the first request is successful, and hence $(n, \vec{b})$ has a left child $(\lceil \frac{n-1}{2} \rceil, \mathbf{odd}(\vec{c}))$ and a right child $(\lfloor \frac{n-1}{2} \rfloor, \mathbf{even}(\vec{c}))$.
This rule applies to all vertices of the tree: For $k > 0$ and binary (infinite) vector $\vec{c}$, a vertex $(k, 0\vec{c})$ in T_{GP} is an internal vertex with one child: $(k-1, \vec{c})$, and a vertex $(k, 1\vec{c})$ has a left child $(\lceil \frac{k-1}{2} \rceil, \mathbf{odd}(\vec{c}))$ and a right child $(\lfloor \frac{k-1}{2} \rfloor, \mathbf{even}(\vec{c}))$. Vertices of the form $(0, \vec{c})$ are leaves. Figure 2 illustrates the execution tree of the GP Algorithm.

The execution $\mathcal{E}_{\text{WU}}(n, \vec{b})$ of the WU algorithm with initial configuration $(n, \vec{b})$ is described by an execution graph $T_{\text{WU}}(n, \vec{b})$ in a similar manner, with one exception: the unique child of a vertex of the form $(k, 0\vec{c})$ for $k > 0$ is $(k, \vec{c})$ (and not $(k-1, \vec{c})$)—reflecting the fact that in a failed request the number of uninformed processors remains unchanged. Note that $T_{\text{WU}}(n, \vec{b})$ may not be a tree, since it may contain vertices (configurations) of the form $(k, 0^{\mathbb{N}})$ which have one outgoing edge which is a self loop (corresponding to the event of infinite sequence of failed requests by a processor). It is not hard to see that $T_{\text{WU}}(n, \vec{b})$ has no other cycles and no other directed infinite paths. Hence $T_{\text{WU}}(n, \vec{b})$ is a finite rooted tree or a finite rooted tree with self loops added to some of its leaves. This latter case correspond to executions in which some processor has an infinite succession of failures.

The following observation is implied by the definitions of T_{GP} and T_{WU}.

Observation 3.2. *For each system configuration $(n, \vec{b})$ it holds that:*

1. *The time complexity of $\mathcal{E}_{\text{GP}}(n, \vec{b})$ is* $\text{height}(T_{\text{GP}}(n, \vec{b}))$.

2. *If $T_{\text{WU}}(n, \vec{b})$ contains a self loop, then the time complexity of $\mathcal{E}_{\text{WU}}(n, \vec{b})$ is infinite. The time complexity of $\mathcal{E}_{\text{WU}}(n, \vec{b})$ is* $\text{height}(T_{\text{WU}}(n, \vec{b}))$*, otherwise.*

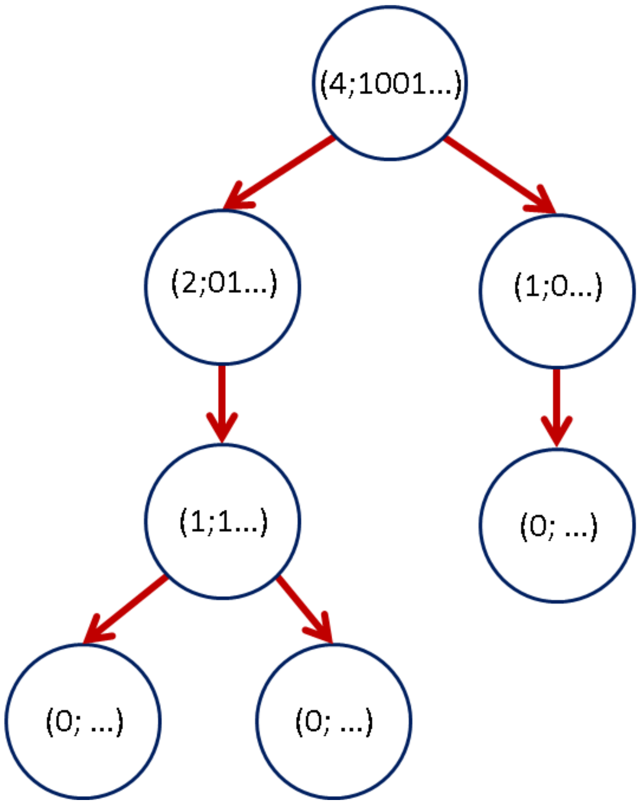

Figure 2: $T_{\mathrm{GP}}(4;1001...)$**:** This tree describes the execution $\mathcal{E}_{\mathrm{GP}}(4;1001\ldots)$ of the GP algorithm for 5 processors and a failure pattern (1001...). Each vertex is a system configuration $(\boldsymbol{k};\vec{b}\,)$, where k is the number of processors to which the rumor need to be delivered, and $\vec{b}$ is the corresponding failure pattern.

3.3.2 Coupling the Models

Here and in the remainder of the paper we abbreviate $h_{GP}(n,\vec{b}\,) = \mathrm{height}(T_{\mathrm{GP}}(n,\vec{b}\,))$ and $h_{WU}(n,\vec{b}\,) = \mathrm{height}(T_{\mathrm{WU}}(n,\vec{b}\,))$.

The main coupling argument is the following lemma, whose inductive proof makes use of the fact that both functions h_{GP} and h_{WU} are monotone increasing in their first argument (i.e., the number of processors to be informed).

Lemma 3.3. *For each system configuration $(n,\vec{b}\,)$ it holds that $h_{GP}(n,\vec{b}\,) \leq h_{WU}(n,\vec{b}\,)$.*

For proving Lemma 3.3, we first observe that both h_{GP} and h_{WU} are monotone increasing in the number of uninformed processors.

Lemma 3.4. *Let $h \in \{h_{GP}, h_{WU}\}$. The function h is monotone increasing in its first argument.*

This lemma follows immediately from the observation that for all $\vec{b}$ and n, $T_{\mathrm{GP}}(n,\vec{b}\,)$ is isomorphic to a proper subtree of $T_{\mathrm{GP}}(n+1,\vec{b}\,)$, and $T_{\mathrm{WU}}(n,\vec{b}\,)$ is isomorphic to a proper subgraph of $T_{\mathrm{WU}}(n+1,\vec{b}\,)$.

We are now ready to prove the main coupling argument, Lemma 3.3.

Proof of Lemma 3.3. For $n = 0$ and for all vectors $\vec{b} \in \{0,1\}^{\mathbb{N}}$ we have

$$h_{GP}(0,\vec{b}\,) = 0 = h_{WU}(0,\vec{b}\,)\,.$$

For the all-zeros vector $\vec{b} = \vec{0}$ and for all $n > 0$ it holds that

$$h_{GP}(n, \vec{0}) = n < \infty = h_{WU}(n, \vec{0}) .$$

We proceed by induction on n, assuming $\vec{b} \neq \vec{0}$. Let b_k be the first non-zero element in $\vec{b}$ (for some $k \geq 1$). It follows that

$$h_{GP}(n, \vec{b}) = k + \max\{h_{GP}\left(\lceil \tfrac{n-k}{2} \rceil, \mathbf{odd}(b_{k+1}, ...)\right), h_{GP}\left(\lfloor \tfrac{n-k}{2} \rfloor, \mathbf{even}(b_{k+1}, ...)\right)\} ,$$

which, by induction hypothesis, can be bounded from above by

$$k + \max\{h_{WU}\left(\lceil \tfrac{n-k}{2} \rceil, \mathbf{odd}(b_{k+1}, ...)\right), h_{WU}\left(\lfloor \tfrac{n-k}{2} \rfloor, \mathbf{even}(b_{k+1}, ...)\right)\} ,$$

which, by Lemma 3.4, is itself bounded from above by

$$\begin{aligned} &k + \max\{h_{WU}\left(\lceil \tfrac{n-1}{2} \rceil, \mathbf{odd}(b_{k+1}, ...)\right), h_{WU}\left(\lfloor \tfrac{n-1}{2} \rfloor, \mathbf{even}(b_{k+1}, ...)\right)\} \\ =&h_{WU}(n, \vec{b}) . \end{aligned}$$

□

Lemma 3.3 and Observation 3.2 show that, for any initial configuration $(n, \vec{b})$, the execution $\mathcal{E}_{\text{GP}}(n, \vec{b})$ of the GP algorithm is at least as fast as the execution $\mathcal{E}_{\text{WU}}(n, \vec{b})$ of the WU algorithm. This implies that for any probability distribution D on $\{0,1\}^{\mathbb{N}}$, if $\vec{b}$ is sampled from D then $\Pr[h_{GP}(n, \vec{b}) \leq H] \geq \Pr[h_{WU}(n, \vec{b}) \leq H]$. By letting D be the standard distribution on $\{0,1\}^{\mathbb{N}}$ with success probability p, Theorem 3.1 easily implies the following.

Theorem 3.5. *Let $c > 1$ be a constant. The execution time of the* GP *algorithm with success probability $p \in (0,1)$ is at most $\frac{c}{p}(\lceil \log(n-1) \rceil + 1)$, with probability at least* $1 - n \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right)$.

4 Adversarial Failures in the Randomized GP-Protocol

In this section we aim at analyzing adversarial failures. As mentioned in Lemma 2.2, it has been proven in [GP96] that the time complexity of the GP algorithm is at most $f + \lceil \log(n-f) \rceil$ when the number of failures is at most f. This bound is sharp when the first f nodes fail. For $f = \omega(\log n)$, this bound is not satisfactory.

We give a modification of the GP algorithm which is *fault-tolerant* in the sense that no matter which constant fraction of the node fails, every processor receives a contact request within the first $O(\log n)$ rounds, with high probability. This protocol can best be described as a randomized version of the basic GP algorithm.

Our algorithm works as follows. When the rumor is injected at processor 0, this processor picks a permutation $\pi \in S_{n-1}$ uniformly at random. In round one it tries to contact processor $\pi(1)$. If this processor has a failure, processor 0 sends a communication request to processor $\pi(2)$ in round two. Otherwise, i.e., if processor $\pi(1)$ is not failed, processor 0 sends to it the rumor and appends to this rumor the list $\mathbf{even}(\pi(2), \ldots, \pi(n-1))$. Processor 0 keeps the list $\mathbf{odd}(\pi(2), \ldots, \pi(n-1))$ as its own todo-list. The protocol continues as described in Section 2.2. That is, all we have changed in our *randomized version of the* GP *algorithm* is to substitute the list of processor 0—which is $(1, \ldots, n-1)$ in the original GP algorithm—by $(\pi(1), \ldots, \pi(n-1))$, where π is a random permutation of $[n-1]$. We also have to append information on π when transferring the rumor. It is not difficult to see (see Section 4.1 below) that this requires a total number of $\Theta(n \log^2 n)$ bits that are appended to the rumors (compared to $\Theta(n \log n)$ in the GP algorithm). The maximum length of an individual message appendix is n bits.

Here and in the remainder of this section we assume (as in all other parts of this work) that the processor initially holding the rumor, node 0, does not fail. Recall that in our initial node failure model, a processor either is a failed one or it does work throughout the execution.

Before we analyze the time complexity of the randomized GP algorithm, let us briefly discuss its *bit complexity*; i.e., the number of bits needed to encode the lists that are appended to the initial rumor.

4.1 The Bit Complexity of the Randomized GP Algorithm

In a naïve implementation of the randomized GP algorithm, every processor passes to its neighbor the list of nodes to be informed by that processor. As described above, in such an implementation, node 0 would pass to node $\pi(1)$ the list $\mathbf{even}(\pi(2), \ldots, \pi(n-1))$ of length smaller than $n/2$. This requires $O(n \log n)$ bits to be appended to the initial rumor. Since the length of the list halves with each successful communication request, in every level of the execution tree the total number of bits that need to be communicated is $O(n \log n)$: We say that a processor state is on the tth *level* if on its unique path to the root exactly t successful calls have happened. For $t \leq \log(n-1)$, in the tth level, there are at most 2^t informed processors, all of which send the rumor to their descendants. Each such processor needs to append a list of length at most $n/2^{t+1}$. This makes a total number of $O(n \log n)$ additional bits that need to be communicated on the tth level. Since there are $O(\log n)$ levels in total, the total bit complexity of this implementation is $O(n \log^2 n)$.

Another implementation of the randomized GP algorithm with the same (asymptotic) bit complexity but a smaller *maximal* appendix is the following. If a processor needs to communicate to its neighbor a list $L = (j_1, \ldots, j_k)$ of length $k > n/\log n$, it appends to the rumor the *incidence list* of L; i.e. a 0/1 vector x of length $n-1$ with $x_i = 1$ if $i \in \{j_1, \ldots, j_k\}$ and $x_i = 0$ otherwise. If a processor has received such a rumor with appended *task list* x, it creates a random permutation π_x of the indices $\{i \mid x_i = 1\}$. It then proceeds as usual, trying to spread the rumor to processor $\pi_x(1)$ in the next round. If less than $n/\log n$ indices need to be communicated, it is cheaper to pass the list itself. It is easily verified, using similar arguments as above, that this implementation yields a total bit complexity of $O(n \log^2 n)$. However, the length of the longest appendix is just linear in n.

In Section 5 we describe an alternative algorithm in which the maximal size of a message appendix is in the order of $\log n$ bits.

4.2 The Time Complexity of the Randomized GP Algorithm

For bounding the time complexity of the randomized GP algorithm we first show that $h_{GP}(n, \vec{b})$, the time complexity of this algorithm for given n and $\vec{b}$, is monotone decreasing in the failure pattern $\vec{b}$, according to the following natural partial order on binary sequences: $(b_1, b_2, \ldots) \leq (c_1, c_2, \ldots)$ if and only if for all $i \in \mathbb{N}$ we have $b_i \leq c_i$.

Lemma 4.1. *The function $h_{GP}(\cdot, \cdot) : \mathbb{N}_0 \times \{0,1\}^{\mathbb{N}} \to \mathbb{R}$ is monotone decreasing in its second argument. That is, for any failure pattern $\vec{b}$, replacing failed processors by non-faulty ones cannot increase the time complexity.*

The proof of Lemma 4.1 uses the following statement, which—informally—says that for each possible failure pattern $\vec{b}$, splitting the rumor spreading at the very beginning between two processors cannot increase the time complexity of the execution.

Lemma 4.2. *For all $n \in \mathbb{N}_0$ and all $\vec{b} \in \{0,1\}^{\mathbb{N}}$ it holds that*

$$h_{GP}(n, \vec{b}) \geq \max\{h_{GP}(\lceil \tfrac{n}{2} \rceil, \boldsymbol{odd}(\vec{b})), h_{GP}(\lfloor \tfrac{n}{2} \rfloor, \boldsymbol{even}(\vec{b}))\}\,. \tag{4}$$

Proof of Lemma 4.2. The proof is by induction on n. The lemma clearly holds for $n = 0$ and $n = 1$. So let $n \geq 2$. Assume first that $\vec{b} = 0\vec{c}$. Then by the definition of T_{GP},

$$h_{GP}(n, 0\vec{c}) = 1 + h_{GP}(n-1, \vec{c}).$$

Using the identities $\lceil \frac{k}{2} \rceil - 1 = \lfloor \frac{k-1}{2} \rfloor$ and $\lfloor \frac{k}{2} \rfloor = \lceil \frac{k-1}{2} \rceil$, we also have

$$h_{GP}(\lceil \tfrac{n}{2} \rceil, \mathbf{odd}(0\vec{c})) = 1 + h_{GP}(\lfloor \tfrac{n-1}{2} \rfloor, \mathbf{even}(\vec{c})) \text{ and}$$
$$h_{GP}(\lfloor \tfrac{n}{2} \rfloor, \mathbf{even}(0\vec{c})) = h_{GP}(\lceil \tfrac{n-1}{2} \rceil, \mathbf{odd}(\vec{c}))\,,$$

which implies (4) by induction.

The case $\vec{b} = 1\vec{c}$ follows along the same lines. To simplify the notations for this case, define $n_1 = \lceil \frac{n-1}{2} \rceil, n_2 = \lfloor \frac{n-1}{2} \rfloor, \vec{d} = \mathbf{odd}(\vec{c})$, and $\vec{e} = \mathbf{even}(\vec{c})$. Then by the definition of T_{GP},

$$h_{GP}(n, 1\vec{c}) = 1 + \max\{h_{GP}(n_1, \vec{d}), h_{GP}(n_2, \vec{e})\}.$$

And the inductive step follows from the inequalities

$$\begin{aligned} h_{GP}(\lceil \tfrac{n}{2} \rceil, \mathbf{odd}(1\vec{c})) &= 1 + \max\{h_{GP}(\lceil \tfrac{n_2}{2} \rceil, \mathbf{odd}(\vec{e})), h_{GP}(\lfloor \tfrac{n_2}{2} \rfloor, \mathbf{even}(\vec{e}))\} \\ &\leq 1 + h_{GP}(n_2, \vec{e}) \text{ (by induction hypothesis),} \end{aligned}$$

and $h_{GP}(\lfloor \frac{n}{2} \rfloor, \mathbf{even}(1\vec{c})) = h_{GP}(n_1, \vec{d})$. □

Proof of Lemma 4.1. The proof is by induction on n. For $n = 0$ we have that $h_{GP}(0, \vec{b}) = 0$ for all $\vec{b}$, and the lemma trivially holds. For the induction step, let $n \geq 0$ and let $\vec{b}, \vec{c}$ be two vectors such that $\vec{b} \leq \vec{c}$. Then $\vec{b} = b_1\vec{d}$ and $\vec{c} = c_1\vec{e}$, where $b_1 \leq c_1$ and $\vec{d} \leq \vec{e}$. If $b_1 = c_1 = 0$ then $\vec{b} = 0\vec{d}, \vec{c} = 0\vec{e}$ and the induction step holds by

$$h_{GP}(n+1, 0\vec{b}) = 1 + h_{GP}(n, \vec{d}) \geq 1 + h_{GP}(n, \vec{e}) = h_{GP}(n+1, 0\vec{c})\,,$$

where the inequality follows from the induction hypothesis. The case $b_1 = c_1 = 1$ is similar and omitted. So we are left with the case $\vec{b} = 0\vec{d}, \vec{c} = 1\vec{e}$ with $\vec{d} \leq \vec{e}$. In this case we have

$$\begin{aligned} h_{GP}(n+1, 0\vec{d}) &= 1 + h_{GP}(n, \vec{d}) \\ &\geq 1 + \max\{h_{GP}(\lceil \tfrac{n}{2} \rceil, \mathbf{odd}(\vec{d})), h_{GP}(\lfloor \tfrac{n}{2} \rfloor, \mathbf{even}(\vec{d}))\} \\ &\geq 1 + \max\{h_{GP}(\lceil \tfrac{n}{2} \rceil, \mathbf{odd}(\vec{e})), h_{GP}(\lfloor \tfrac{n}{2} \rfloor, \mathbf{even}(\vec{e}))\} \\ &= h_{GP}(n+1, 1\vec{e}),\ , \end{aligned}$$

where the first inequality follows from Lemma 4.2, and the latter inequality follows from the induction hypothesis on $\mathbf{odd}(\vec{d}), \mathbf{odd}(\vec{e})$ and on $\mathbf{even}(\vec{d}), \mathbf{even}(\vec{e})$. □

Note: A similar but slightly more involved argument shows that Lemma 4.1 holds also for the function $h_{WU}(\cdot, \cdot)$.

Theorem 4.3. *Let $n \in \mathbb{N} \setminus \{1\}$. Let $\varepsilon = \sqrt{\frac{\ln n}{n-1}}$. Let $f < (n-1)(1-\varepsilon)$ and let $F \subseteq [n-1]$ of size $|F| = f$. Let $p = 1 - \frac{f}{n-1}$. Let $c > 1$ be a constant.*

The probability that the randomized version of the GP *algorithm has time complexity $T \leq \frac{c}{p-\varepsilon}(\lceil \log(n-1) \rceil + 1)$ is at least $1 - \frac{n^3}{n^2-1} \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right)$, even if all processors in F fail.*

As we mentioned after Theorem 3.1, the probability bound is $1 - o(1)$ for c satisfying $\frac{(c-1)^2}{2c\ln(2)} > 1$. The proof of Theorem 4.3 is via a reduction to the random failure model analyzed in Section 3.1. It makes use of several Chernoff bounds and the monotonicity proven in Lemma 4.1. We basically show that the runtime of the randomized protocol is not worse than that of the basic deterministic GP algorithm under the presence of independent random failures. In the latter, we chose the failure probability to be slightly larger than the "fair" ratio $f/(n-1)$, so that, with probability at least $1 - n^{-2}$, more than f nodes are crashed. Combining this with the resulting runtime bound from Theorem 3.5 proves Theorem 4.3.

Proof of Theorem 4.3. It is easy to verify that the randomized GP algorithm with f adversarial failures has the same performance as the original GP algorithm when a random subset of node failures, $R \subseteq [n-1]$ with $R = f$, is selected uniformly. We analyze the latter.

Let $p' := p - \varepsilon$ and $T := \frac{c}{p'}(\lceil \log(n-1) \rceil + 1)$. Let $\vec{b} \in \{0,1\}^{n-1}$ with $|\vec{b}|_0 = f$ be chosen uniformly at random. We need to show that

$$\Pr[h_{GP}(n, \vec{b}) > T] \leq \frac{n^3}{n^2 - 1} \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right).$$

Let $\vec{c} \in \{0,1\}^{n-1}$ be such that $\Pr[\vec{c}_i = 1] = p'$ independently for all $i \in [n-1]$. That is, the probability that $\vec{c}_i = 0$ is $\frac{f}{n-1} + \varepsilon$, for every $i \in [n-1]$. We show

$$\Pr[|\vec{c}|_0 \geq f] \geq 1 - n^{-2}, \tag{5}$$

which can be easily verified by Chernoff's bound: The expected value of $|\vec{c}|_0$ is $f + \varepsilon(n-1)$. By Chernoff's bound, cf. Theorem 2.3, equation (1), we have

$$\begin{aligned} \Pr[|\vec{c}|_0 < f] &= \Pr[|\vec{c}|_0 < \mathrm{E}[|\vec{c}|_0] - \varepsilon(n-1)] \leq \exp\left(-2(\varepsilon(n-1))^2/(n-1)\right) \\ &= \exp\left(-2\varepsilon^2(n-1)\right) = \exp(-2\ln n) = n^{-2}. \end{aligned}$$

Next we argue that

$$\Pr[h_{GP}(n, \vec{b}) \geq T] \leq \Pr[h_{GP}(n, \vec{c}) \geq T \mid |\vec{c}|_0 \geq f]. \tag{6}$$

To verify (6), assume $|\vec{c}|_0 > f$. Sample $k := |\vec{c}|_0 - f$ indices $i_1, \ldots, i_k$ from the 0-positions $\{i \in [n-1] \mid \vec{c}_i = 0\}$ of $\vec{c}$ uniformly at random. Create $\vec{d}$ from $\vec{c}$ by replacing the zeros in positions $i_1, \ldots, i_k$ by ones. Then $\vec{d}$ is uniform in the set $\{\vec{b} \in \{0,1\}^{n-1} \mid |\vec{b}|_0 = f\}$, as is $\vec{b}$. Inequality (6) follows from the latter and the monotonicity of $h_{GP}(n, \vec{b})$ in $\vec{b}$, as stated in Lemma 4.1.

Using this inequality we bound

$$\begin{aligned} \Pr[h_{GP}(n, \vec{b}) > T] \cdot \Pr[|\vec{c}|_0 \geq f] &\leq \Pr[h_{GP}(n, \vec{c}) > T \mid |\vec{c}|_0 \geq f] \cdot \Pr[|\vec{c}|_0 \geq f] \\ &\leq \Pr[h_{GP}(n, \vec{c}) > T \mid |\vec{c}|_0 \geq f] \cdot \Pr[|\vec{c}|_0 \geq f] \\ &\quad + \Pr[h_{GP}(n, \vec{c}) > T \mid |\vec{c}|_0 < f] \cdot \Pr[|\vec{c}|_0 < f] \\ &= \Pr[h_{GP}(n, \vec{c}) > T]. \end{aligned}$$

The latter quantity can be bounded by Theorem 3.5. It shows that the time complexity of the GP algorithm with success rate p' satisfies

$$\Pr[h_{GP}(n, \vec{c}) > T] \leq n \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right).$$

Together with inequality (5), this concludes the proof. □

5 Reducing the Message Size

Building on the results from the previous sections, we now describe an alternative version of the GP algorithm that has a message overhead of only $O(\log n)$ bits per rumor transfer, but that is still robust against adversarial failures. This is achieved at the price of adding a preprocessing phase and extra storage space to each of the processors.

More precisely, we show that for $t \in O(nh(n)/\log n)$, $h \in \omega(1)$ being an arbitrary function tending to infinity, there are t permutations such that, no matter which constant fraction of the processors fail, the probability that a permutation chosen uniformly at random out of the t yields a runtime that is greater than $c \log n$ is $o(1)$ (both the constant c and the $o(1)$ failure probability will be made precise below). The algorithm is based on storing these t permutations at each of the processors.

Let $\{\pi^1, \ldots, \pi^t\} \subseteq S_{n-1}$ be the stored permutations. Upon receiving a rumor, processor 0 chooses at random an index $r \in [t]$. The algorithm now is the following minor modification of the original GP algorithm: At each round, a processor i which holds a nonempty list $(j_1, \ldots, j_k)$ sends a communication request to processor $\pi^r(j_1)$, and deletes j_1 from its list. If $\pi^r(j_1)$ is non-faulty, then i sends it the rumor appended with (a) the index r, (b) the value j_3, (c) the length $\lfloor \frac{k-1}{2} \rfloor$ of the list to be informed by processor $\pi^r(j_1)$, and (d) the exponent m of the arithmetic progression $\mathbf{even}(j_2, \ldots, j_k)$. Processor $\pi^r(j_1)$ starts the next round with the list $\mathbf{even}(j_2, \ldots, j_k)$, and processor i starts it with the list $\mathbf{odd}(j_2, \ldots, j_k)$.

To pass information (b)–(d), $3(\lceil \log n \rceil + 1)$ bits suffice. To pass information (a), $\lceil \log t \rceil + 1$ bits are needed. Thus, for $t \in O(n^d)$ for a constant d, the overall number of bits that need to be appended to the rumor is $O(\log n)$.

As mentioned above, the main goal of this section is to show (Theorem 5.2) that for $t \in \omega(n/\log n)$ and suitably chosen permutations $\pi^1, \ldots, \pi^t$ this protocol, with high probability, is robust against adversarial failures.

Definition 5.1. *We call the* $\mathrm{GP}(\pi^1, \ldots, \pi^t)$ *algorithm described above* (f, r, T)-safe *if, for each possible failure pattern* $F \subseteq [n-1]$ *with* $|F| = f$*, it holds that with probability at least* r *the runtime of the protocol* $\mathrm{GP}(\pi^1, \ldots, \pi^t)$ *with failure pattern* F *is at most* T*.*[5]

Interestingly, for any constant $d < 1$ and for $t \in \omega(n/\log n)$, t randomly chosen permutations $\pi^1, \ldots, \pi^t$ are $(dn, 1 - o(1), O(\log n))$-safe, with high probability.

Theorem 5.2. *Let* $t \in \omega(n/\log n)$*. Let* $\pi^1, \ldots, \pi^t$ *be taken from* S_{n-1} *independently and uniformly at random. Let* $\varepsilon := \sqrt{\ln n/(n-1)}$*,* $f < (n-1)(1-\varepsilon)$*, and* $p := 1 - \frac{f}{n-1}$*.*

There are $c = c(n) \leq 6 + o(1)$ *and* $\delta = \delta(n)$ *with* $\lim_{n\to\infty} \delta(n) = 0$*, such that the probability that* $\mathrm{GP}(\pi^1, \ldots, \pi^t)$ *is* $(f, 1-\delta, \frac{c}{p-\varepsilon}(\lceil \log(n-1) \rceil + 1))$*-safe is* $1 - o(n^{-1})$*.*

The proof of Theorem 5.2 is based on Theorem 4.3: By that theorem we know that, for a fixed failure set F and a random permutation π, the probability that the randomized GP algorithm along permutation π and failure set F exceeds the desired runtime T is less than n^{-c}. Based on this, we show that the fraction of $\omega(n/\log n)$ randomly chosen permutations that exceed runtime T is less than δ, with probability exponentially small in n. A union bound over all possible failure patterns F concludes the proof.

Proof of Theorem 5.2. By the assumption on t we have a function $\delta = \delta(n)$ satisfying $\delta \cdot t > 2n/\log(n)$ and $\lim_{n\to\infty} \delta(n) = 0$. Select such δ satisfying also $\delta(n) > e/n$.

[5]The probability statement in this definition is with respect to the random choice of the permutation index $i \in [t]$.

We now define $c = c(n) > 1$. Fix—for the moment—some failure set $F \subseteq [n-1]$ of size $|F| = f$. For given $\pi^1 \dots \pi^t$, let T^i be the time complexity of the GP algorithm along permutation π^i if all processors in F fail. Since the index i is chosen uniformly, the probability that the runtime of the GP algorithm with failure set F exceeds

$$T := \frac{c}{p-\varepsilon} \left(\lceil \log(n-1) \rceil + 1\right)$$

is the fraction of indices $i \in [t]$ with $T^i > T$. By Theorem 4.3 we know that, for a random permutation σ of $[n-1]$, the probability that the runtime of the GP algorithm along permutation σ and failure set F exceeds T is at most

$$q := \frac{n^3}{n^2-1} \exp\left(-\frac{(c-1)^2}{2c}(\lceil \log(n-1) \rceil - 1)\right) = n \cdot \exp\left(-\frac{(c-1)^2}{2c} \log n(1+o(1))\right) .$$

We select $c = c(n)$ such that $q < n^{-2}$. Using the fact that $\log n = \ln(n)/\ln(2) \approx 1.44 \ln(n)$, it can be easily verified that c can be chosen such that $c \leq 6 + o(1)$ as claimed.

We now show that for this value of c, the probability that the fraction of indices i with $T^i > T$ is larger than δ, is exponentially small. To obtain the statement of the theorem, we will then do a union bound over all possible choices of F.

The probability that for the fixed F and randomly chosen permutations $\sigma^1, \dots, \sigma^t$ at least t_0 of them yield a runtime exceeding T is at most

$$\sum_{j=t_0}^{t} \binom{t}{j} q^j (1-q)^{t-j} < \sum_{j=t_0}^{t} \frac{t^j}{j!} q^j < \sum_{j=t_0}^{t} \left(\frac{etq}{j}\right)^j, \tag{7}$$

where the right inequality is by Stirling approximation for $j!$.

We now set $t_0 := \lceil \delta t \rceil$, which, by definition of δ is at least $2n/\log(n)$. By the definition of δ and c, the sum above is dominated by the sum of the geometric progression $(a^{t_0}, a^{t_0+1}, \dots, a^t)$ for $a \geq eqt/t_0 \approx eq\delta$. Since $\delta > e/n$, the value of a can be chosen to be less than $1/n$, for sufficiently large n. Thus, the first element in this progression, a^{t_0}, is smaller than $n^{-2n/\log(n)} = 2^{-2n}$. Hence, the sum in (7) is smaller than $2 \cdot 2^{-2n}$.

We now do a union bound over all possible choices of F. There are at most $\binom{n}{f} < 2^n$ different choices. Therefore, the probability that for t randomly chosen permutations there exits a choice of F such that the corresponding runtime is larger than T, is smaller than $2^n \cdot 2 \cdot 2^{-2n} = 2 \cdot 2^{-n}$. $\square$

Note that the definition of δ in the above proof implies that if $t(n) > 2n^2/\log n$ then δ in Theorem 5.2 can be set to $\delta(n) = e/n$.

Remark: The preprocessing phase, and the extra storage space, needed to generate and store the random permutations, can be eliminated if there is an efficient construction of a set S of $O(n^k)$ permutations (for some small k) which, for any $p < 1$, and for any choice of pn failures, the runtime of GP on a random permutation from S is w.h.p. $O(\log n/p)$. This is particularly appealing in environments where processors may be added or removed from the networks. The existence of such a construction remains an interesting open problem.

6 Conclusions and Future Work

We have studied fault-tolerant rumor spreading algorithms in the initial failures model, where, for arbitrary $p \in (0,1)$, an arbitrary set of pn processors may fail. The algorithms are based on introducing randomization to the elegant whispering algorithm of [GP96]. They have minimal message complexity and asymptotically optimal time complexity, do not require synchronization or activation of uninformed processors, do not need to assume an a priori bound on the number of faulty processors, and do not need an opening phase.

We proved that the time complexity of the GP algorithm in the presence of random initial failures is asymptotically optimal, i.e., $O(\log n)$. The analysis is based on a new *random wakeup model* and a novel coupling technique, which could be of independent interest. To deal with adversarial failures, we have proposed a *randomized version* of the GP algorithm. While the randomized GP algorithm achieves best possible message complexity and asymptotically optimal time complexity, it has an overhead of a total number of $O(n \log^2 n)$ bits that need to be communicated. We have shown that, using a preprocessing step for storing $O(nh(n)/\log n)$ random permutations at the processors ($h \in \omega(1)$ being an arbitrary function tending to infinity), we can decrease this overhead to $O(n \log n)$ bits. In this protocol, the maximum number of bits that need to be appended to any message is in the order of $\log n$. The preprocessing phase of this protocol could be eliminated if a set S of a polynomial number of permutations was constructed by an efficient deterministic algorithm such that for any $p < 1$ and for any set F of $(1-p)n$ faulty processors, with probability $1 - o(1)$, the runtime of the GP protocol on a random permutation from S is $O(\log(n))$.

Acknowledgments

Carola Doerr is supported by a Feodor Lynen postdoctoral research fellowship of the Alexander von Humboldt Foundation and by the Agence Nationale de la Recherche under the project ANR-09-JCJC-0067-01.

Shlomo Moran is supported by the Bernard Elkin Chair in Computer Science.

References

[AD11] Anne Auger and Benjamin Doerr, *Theory of randomized search heuristics*, World Scientific, 2011.

[Bil95] Patrick Billingsley, *Probability and measure*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, Inc., New York, 1995.

[CHHKM12] Keren Censor-Hillel, Bernhard Haeupler, Jonathan A. Kelner, and Petar Maymounkov, *Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance*, Proc. of the ACM Symposium on Theory of Computing (STOC), ACM, 2012, pp. 961–970.

[DF11a] Benjamin Doerr and Mahmoud Fouz, *Asymptotically optimal rumor spreading*, Proc. of the International Colloquium on Automata, Languages, and Programming (ICALP), Springer, 2011, pp. 502–513.

[DF11b] Benjamin Doerr and Mahmoud Fouz, *Quasi-random rumor spreading: Reducing randomness can be costly*, Information Processing Letters **111** (2011), no. 5, 227–230.

[DGH$^+$88] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry, *Epidemic algorithms for replicated database maintenance*, Operating Systems Review **22** (1988), no. 1, 8–32.

[DHL09] Benjamin Doerr, Anna Huber, and Ariel Levavi, *Strong robustness of randomized rumor spreading protocols*, Proc. of the International Symposium on Algorithms and Computation (ISAAC), Springer, 2009, pp. 812–821.

[DP00] Krzysztof Diks and Andrzej Pelc, *Optimal adaptive broadcasting with a bounded fraction of faulty nodes*, Algorithmica **28** (2000), no. 1, 37–50.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi, *Concentration of measure for the analysis of randomised algorithms*, Cambridge University Press, 2009.

[ES09] Robert Elsässer and Thomas Sauerwald, *On the runtime and robustness of randomized broadcasting*, Theoretical Computer Science **410** (2009), no. 36, 3414–3427.

[FG85] Alan M. Frieze and Geoffrey R. Grimmett, *The shortest-path problem for graphs with random arc-lengths*, Discrete Applied Mathematics **10** (1985), 57–77.

[GP96] Leszek Gasieniec and Andrzej Pelc, *Adaptive broadcasting with faulty nodes*, Parallel Computing **22** (1996), no. 6, 903–912, preliminary version in http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.3838.

[KSSV00] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking, *Randomized rumor spreading*, Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, 2000, pp. 565–574.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica **8** (1988), no. 3, 261–277 (English).

[MAS06] Damon Mosk-Aoyama and Devavrat Shah, *Computing separable functions via gossip*, Proc. of the ACM-SIGOPT Principles of Distributed Computing (PODC), ACM, 2006, pp. 113–122.

[Pit87] Boris Pittel, *On spreading a rumor*, SIAM Journal on Applied Mathematics **47** (1987), no. 1, 213–223.

[RS98] Martin Raab and Angelika Steger, *"Balls into bins" - a simple and tight analysis*, Proc. of the International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM), Springer, 1998, pp. 159–170.

[TKM89] Gadi Taubenfeld, Shmuel Katz, and Shlomo Moran, *Initial failures in distributed computations*, International Journal of Parallel Programming **18** (1989), no. 4, 255–276.

[Upf94] E. Upfal, *Tolerating a linear number of faults in networks of bounded degree*, Information and Computation **115** (1994), no. 2, 312 – 320.

[Yao77] Andrew Chi-Chih Yao, *Probabilistic computations: Toward a unified measure of complexity (extended abstract)*, Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, 1977, pp. 222–227.