Gossiping to Reach Consensus

Bogdan S. Chlebus
Computer Science and Engineering
University of Colorado at Denver
and
Instytut Informatyki
Uniwersytet Warszawski
Bogdan.Chlebus@cudenver.edu

Dariusz R. Kowalski
Instytut Informatyki
Uniwersytet Warszawski
and
Département d'Informatique
Université du Québec à Hull
darek@mimuw.edu.pl

ABSTRACT

We consider the problem of gossiping when dynamic node crashes are controlled by adaptive adversaries. We develop gossiping algorithms which are efficient with respect to both the time and communication measured as the number of point-to-point messages. If the adversary is allowed to fail up to t nodes, among the total of n, where additionally $n-t = \Omega(n/\text{polylog } n)$, then one among our algorithms completes gossiping in time $\mathcal{O}(\log^2 t)$ and with $\mathcal{O}(n \text{ polylog } t)$ messages. We prove a lower bound which states that the time has to be at least $\Omega\left(\frac{\log n}{\log(n \log n) - \log t}\right)$ if the communication is restricted to be $\mathcal{O}(n \text{ polylog } n)$.

We also show that one can solve efficiently a more demanding consensus problem with crash failures by resorting to one of our gossiping algorithms. If the adversary is allowed to fail t nodes, where $n - t = \Omega(n/\text{polylog } n)$, we obtain a time-optimal solution that is away from the communication optimality by at most a polylogarithmic factor.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Networks Architecture and Design*; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; F.2.3 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Tradeoffs between Complexity Measures*

General Terms

theory, algorithms, reliability

Keywords

distributed algorithm, processor failures, adaptive adversary, gossiping, consensus, lower bound

1. INTRODUCTION

Many distributed applications are based on collective updating of data that originate at the nodes of a communication network. The problem of gathering distributed information is known as the *gossip problem:* in the beginning each processor has its distinct piece of information, called a rumor, the goal is to make every processor know all the rumors.

We would like to perform gossiping not only quickly but also with as few node-to-node transmissions as possible. This is especially challenging when the network is prone to failures of components. We consider the case when the nodes may crash. Then it may be not always feasible to collect the input rumor of a node v that has failed, since all the nodes that have learned the rumor of v also might have failed in the course of an execution of a gossiping algorithm. We consider the task of gossiping to be completed when every node has learned the following about each node v: either the rumor of v or that v has failed. Note that if a processor fails then some of the nodes may get to know its input rumor but others may only learn that it has failed.

In the *consensus problem,* the processors need to agree upon a common value, among their input ones, while the processors are subject to failures. This is related to the gossip problem, in that once each processor knows all the input values then the decision can be taken by applying some simple rule. For instance, the processors may want to agree upon the maximum value. However, just running a gossiping algorithm is not enough: it may happen that the maximum value is known only to some of the processors, if the original holder of the value crashed in the course of trying to disseminate it.

We consider a synchronous distributed environment, in which processors communicate among themselves by exchanging messages. The efficiency of algorithms is measured with respect to two performance metrics: time and communication. The communication is gauged as the number of point-to-point messages.

Our results. An important novelty in our approach to the gossip problem is to consider adaptive adversaries that are able to fail processors. In the prior research on the gossip problem in failure-prone environments, either link failures or processor failures controlled by oblivious adversaries have been considered, see the survey by Pelc [33].

Let n be the number of processors and t the number of failures that the algorithm is to tolerate.

I. We develop gossiping algorithms that are efficient with respect to both the time and the number of point-to-point messages sent. One of these algorithms runs in time $\mathcal{O}(\log^2 t)$ and sends $\mathcal{O}(n \text{ polylog } t)$ point-to-point messages, if only $n - t = \Omega(n/\text{polylog } n))$. We also give a general theorem that covers stronger adversaries. In particular, if the adversary can fail all but one processor then the time is $\mathcal{O}(\log^2 n)$ and communication is $\mathcal{O}(n^{1.77})$, which is subquadratic in n.

II. We also prove a lower bound on gossiping, to gauge how far from optimality our gossiping algorithm is when both time and communication are taken into account.

We show that, if communication is $\mathcal{O}(n \text{ polylog } n)$, then the time to achieve gossiping against t failures has to be at least $\Omega(\log n/(\log(n \log n) - \log t))$. This proves that our gossiping algorithm operating in time $\mathcal{O}(\log^2 t)$ misses the time optimality by at most a logarithmic factor, in the class of algorithms that send the number of messages differing from linear by a polylogarithmic factor.

III. Our main result is a consensus algorithm that is able to handle crash failures, and which is optimal with respect to time *and* efficient with respect to communication. The consensus algorithm uses the gossiping algorithm as a building block.

We present an algorithm tolerating t crashes and solving the consensus problem in the optimal time $\mathcal{O}(t)$ and with $\mathcal{O}(n \text{ polylog } t)$ messages, if only $n - t$ is in $\Omega(n/\text{polylog } n)$. The most efficient, with respect to communication, previously known algorithm, among time-optimal ones, used $\mathcal{O}(n + tn^{\varepsilon})$ messages. It has also been shown before how to solve consensus with $\mathcal{O}(n)$ messages, but the best known algorithms achieving this require time $\Omega(n^{1+\varepsilon})$, for any fixed $\varepsilon > 0$.

Background. One may consider dissemination of information in a network similarly as a spread of rumor or of an infectious disease in a society, which has been studied in applied mathematics [7]. Demers *et al.* [13] introduced so-called epidemic algorithms for updating data bases, in which a node regularly chooses other nodes at random and transmits the rumors. Such randomized epidemic algorithms have been systematically studied by Karp, Schindelhauer, Shenker and Vöcking [26]. The problem of exchange of information when machines do not initially know each other was considered by Harchol-Balter, Leighton and Lewin [25], in their solution the nodes learn about each other in the course of gossiping. Gossip-style algorithm in which nodes learn about the nearest resource location was given by Kempe, Kleinberg and Demers [27]. Application of gossiping to gathering information about occurrences of failures was done by van Renesse, Minsky and Hayden [37].

The asynchronous version of gossiping, known as *collect*, was first abstracted by Saks, Shavit, and Woll [35]. The goal is to have all the processes learn all the register values. Computation is asynchronous, with the adversary controlling timing of the processors. Ajtai, Aspnes, Dwork and Waarts [1] showed that the problem can be solved deterministically with work $\mathcal{O}(n^{3/2} \log n)$, by an adaptation of the algorithm of Anderson and Woll [3]. Aspnes and Hurwood [6] developed a randomized algorithm achieving work $\mathcal{O}(n \log^3 n)$ with high probability. A lower bound of $\Omega(n \log n)$ for this problem is given in [35].

The problem of consensus was introduced by Pease, Shostak and Lamport [32]. They showed [29, 32] that the number t of faulty processors needs to be smaller than $n/3$ for a solution to exist, assuming there is synchrony. The condition $3t < n$ is both sufficient and necessary. The faults required for this result to hold are of the malicious and unrestricted character, called Byzantine [30].

Fisher, Lynch and Paterson [19] showed that the problem is unsolvable in an asynchronous setting, even with only one crash failure. Randomization can make solutions possible in an asynchronous setting, as was observed by Ben-Or [9] and Rabin [34]. The issue of the use of randomization in consensus protocols has been intensively studied, in particular by Aspnes [4, 5], Bar-Joseph and Ben-Or [8] and Goldreich and Petrank [22].

For the relevance of the consensus problem to fault-tolerant broadcast and other communication problems see the paper by Hadzilacos and Toueg [24]. Fisher and Lynch [18] showed that a (synchronous) solution requires $t + 1$ rounds, when t is the tolerable number of failures. Garay and Moses [21] developed an algorithm with polynomial-size messages for $n > 3t$ processors in $t + 1$ rounds. The message complexity of consensus when no failures actually happen was studied by Amdur, Weber, and Hadzilacos [2] and Hadzilacos and Halpern [23].

Dolev and Reischuk [15] studied the message complexity of the consensus problem in the case of malicious faults. They distinguished between the general case of Byzantine faults and the less demanding situation when some (cryptographic) authentication mechanism is available, which makes forging signatures of forwarded messages impossible. They showed a lower bound $\Omega(nt)$ on the number of signatures for any algorithm using authentication, which is also a lower bound on the total number of messages for any protocol without authentication. They also showed that any algorithm with authentication needs to send $\Omega(n + t^2)$ messages, and that achieving these many messages is possible. This shows that in the case of malicious adversaries the required number of messages can be as high as quadratic in n.

For crash failures, which are benign compared to Byzantine, the issue of message complexity of consensus is radically different. For a long time only the trivial linear lower bound $\Omega(n)$ on the number of messages was known, and the issue of its optimality was open. The first solutions, like the one given by Lamport and Fisher [28], had $\mathcal{O}(n^3)$ message complexity. Bracha [10] showed how to achieve a subquadratic message complexity. Dwork, Halpern and Waarts [16] found a solution with $\mathcal{O}(n \log n)$ messages but with exponential time. Finally, Galil, Mayer and Yung [20] developed an algorithm with $\mathcal{O}(n)$ messages, thus showing that this amount of messages is optimal. The drawback of their solution is that it runs in overlinear time $\mathcal{O}(n^{1+\varepsilon})$, for any $0 < \varepsilon < 1$.

A solution to consensus has the early-stopping property if the running time is $\mathcal{O}(t + 1)$ where t is *the actual number of failures in an execution,* this time $\mathcal{O}(t + 1)$ is optimal. Lamport and Fischer [28] gave an early stopping algorithm with $\mathcal{O}((t + 1)n^2)$ messages. Chandra and Toueg [11] improved this to $\mathcal{O}((t+1)n)$. Galil, Mayer and Yung [20] found an early-stopping solution with $\mathcal{O}(n + tn^{\varepsilon})$ communication complexity, for any $0 < \varepsilon < 1$.

Upfal [36] showed how an almost everywhere agreement can be achieved with a linear number of faults in networks of a constant degree, extending the result by Dwork, Peleg, Pippenger and Upfal [17]. His approach was then further extended by Chlebus, Gąsieniec, Kowalski and Shvartsman [12] in their work on the problem of performing independent and similar tasks.

2. TECHNICAL PRELIMINARIES

There are n processors, each with a unique identifier in the interval $[1..n]$. The computations are synchronous, with the processors having access to a global clock. Processes communicate by message passing. Each processor can send a message to any subset of processors in one step, it is delivered to each recipient by the end of this step. We consider two measures of performance: time and communication, where communication means the total number of point-to-point messages sent by termination.

Failures. Processors may fail by crashing. We do not assume failures to be clean: if a processor fails, while attempting to multicast a message, then some of the recipients may receive the message and some not. Except for that the communication is reliable, and messages are not lost or corrupted, while in transit. Failure patterns are considered in the framework of a suitable adversarial model. Our adversaries are adaptive, in that they make decisions which processors to fail on-line, usually subject to some quantitative restrictions. An adversary is *size bounded* if there is an upper bound on the number of failures it is allowed to incur; usually we denote this bound by t, then the adversary is called *t-bounded*. An adversary is *linearly-bounded* if it is cn-bounded, for a constant $0 < c < 1$. The *unbounded* adversary can fail up to $n-1$ processors. An algorithm is *resilient against a given adversary* if it is correct if run against it. It is *t-resilient* if it is correct against a t-bounded adversary, and in the case of $t = n-1$ such an algorithm is simply called *resilient*.

Graphs. Graphs $G = (V, E)$ used in this paper are always undirected. The set V of nodes is a subset of all the processors. The edges in E are used to determine pairs of nodes that communicate directly. The kth power G^k of G is defined as follows: it has V as its set of nodes, two nodes are connected by an edge in G^k iff there is a path between them in G of length at most k. We use special graphs G and G^k to have communication patterns that allow to save on the number of messages. The underlying graph G is a constructive expander, as defined by Lubotzky, Phillips and Sarnak [30]. For a positive integer m, the network $G(m)$ has m nodes and it simulates such an expander graph of $\Theta(m)$ nodes (see [12, 30]). The idea of using powers of expander graphs to amplify their expansion was first suggested by Margulis [31], and used for communication purposes by Diks and Pelc [14].

The graph $G(m)$ has a constant bound on the node degrees, we denote it by Δ_0, and its diameter is $\mathcal{O}(\log m)$. We will use the following property of graphs $G(m)$, we call it *property $\mathcal{R}(m,t)$,* where $t < m \le 2t$:

> There is a function P, which, for each subgraph $R \subseteq G(m)$ of size at least $m-t$, defines a subgraph $P(R) \subseteq G(n)$ such that the following hold:

1. $P(R) \subseteq R$.
2. $|P(R)| = |R|/7$.
3. $Diameter\big(P(R)\big) \le 30 \log m + 2$.
4. If $R_1 \subseteq R_2$ then $P(R_1) \subseteq P(R_2)$.

Note that $P(R)$ is connected, even if R is not, since its diameter is finite. Upfal [36] proved that if $m - t \ge \frac{71}{72}m$ then graph $G(m)$ has the property $\mathcal{R}(m,t)$. This was later extended by Chlebus, Gąsieniec, Kowalski and Shvartsman [12], who showed that, for *any* parameter t such that $t < m \le 2t$, a graph obtained from $G(m)$ by taking a sufficiently large power has property $\mathcal{R}(m,t)$. This result is used in this paper in the following form:

FACT 1 ([12]). *For every $t < m \le 2t$ there exists a positive integer ℓ such that graph $G(m)^\ell$ has property $\mathcal{R}(m,t)$. Moreover, the maximum degree Δ of a graph $G(m)^\ell$ is bounded by $\mathcal{O}\Big(\frac{m}{m-t}\Big)^{2\log_\lambda \Delta_0}$, for constants Δ_0 and λ, where $\lambda > 1$ depends on the expansion of $G(m)$.* □

FACT 2 ([36, 14, 12]). *The following bound holds:* $(1 + 2\log_\lambda \Delta_0)^{-1} > 0.23$. □

From now on we consider the specific value $m = \min\{n, 2t\}$. We define graph $G(n,t)$ as follows. All the processors serve as vertices. Edges are of two kinds. First we take the edges of a graph isomorphic with $G(m)^\ell$, whose nodes are the processors with ids in the range 1 through m, and ℓ is such that $G(m)^\ell$ has property $\mathcal{R}(m,t)$, as guaranteed by Fact 1. Secondly, if $m < n$ then we add all connections of the form (p, q), where $p \le m$ and $q \equiv p \mod m$. Graph $G(n,t)$ is called the *communication graph.* This graph may evolve in the course of an execution of a gossiping algorithm: if a processor fails then it is no longer a node.

3. GOSSIPING ALGORITHM

In this section we describe a gossiping algorithm called Algorithm_Gossip. The following parameters are hardwired into the algorithm: the number n of processors, the maximum number t of failures, the graph $G(n,t)$, an integer *termination threshold* T, and a family $\Pi = \{\pi_p\}_{1\le p\le m}$ of *local permutations*. Only the parameters T and Π have not been presented yet, their role will be described in the specification of algorithm, and their properties related to the efficiency of the algorithm will be discussed in Section 4.

We say that a processor p *heard about processor* q if either p knows the original input rumor of q or it knows that q has failed.

Categories of processors. We partition the set of all the processors into m *groups*, where $m = \min\{n, 2t\}$. Group $D(i)$, for $1 \le i \le m$, is defined to consist of processors q such that $q \equiv i \mod m$. Groups are of balanced size, each has either $\lfloor n/m \rfloor$ or $\lfloor n/m \rfloor + 1$ processors. The processor with the smallest id in $D(i)$ is called its *leader*. The subgraph of $G(n,t)$ induced by the leaders is isomorphic to $G(m)^\ell$.

Each leader starts an execution as a *collector*. Collectors actively seek information about the rumors of the other processors, by sending direct inquiries to them. After a collector has heard about all the processors then it becomes a *disseminator*. These processors disseminate their knowledge by sending out their local state. The processors that are not leaders are called *ordinary*. The ordinary processors are most passive among all the categories of processors. They do not seek information on their own, relying instead on the initiative of the leaders. An ordinary processor just waits till messages come to it. If such a message is an inquiry from a collector then the ordinary processor replies to

Receiving Step: Receive messages

Computation Step:

a. Update the local arrays
b. If p is a collector that has heard about all the processors then become a disseminator

Sending Step:

a. Send a graph message to each neighbor q of p in the communication graph that is active according to p
b. Send a reply message to each collector q from which a range message was received in the receiving step of this phase
c. If p is a collector and q is in the first chunk with a processor about which p has not heard yet then send a range message to q
d. If p is a disseminator and q is in the first chunk with a processor that needs to be notified by p then send a range message to q

Figure 1: Regular phase of Algorithm_Gossip. Code for processor p. The ordering of chunks, that processor p uses if it is a leader, is determined by the permutation π_p.

Receiving Step: Receive messages

Computation Step: Update the local arrays

Sending Step: For each processor q:

a. If p has not heard about processor q yet then send a last-resort message to q
b. If a last-resort message was received from q in the receiving step of this phase then send a reply message to q

Figure 2: Ending phase of Algorithm_Gossip. Code for processor p.

it by sending its local state. If the message is a summary of the knowledge of a disseminator, then it makes the ordinary processor sufficiently informed.

Local state. Initially each processor p knows only its id p and its input $rumor_p$. To store incoming information, processor p maintains the following arrays: $\texttt{Rumors}_p$, $\texttt{Active}_p$ and $\texttt{Pending}_p$, each of size n. All these arrays are initialized to zeros. If an entry if the array is not equal to the initial zero then it is some information with a special meaning that has been stored. Given an array $\texttt{X}_p$ of processor p, we denote by $\texttt{X}_p[q]$ the contents of its qth location.

The array Rumor is used to store all the rumors that a processor knows. In the very beginning p sets $\texttt{Rumors}_p[p]$ to its own input $rumor_p$. After processor p learns some $rumor_q$ it immediately sets $\texttt{Rumors}_p[q]$ to this value.

The array Active is used to store the set of all the crashed nodes that a processor knows about. Once processor p learns that processor q has failed it immediately sets $\texttt{Active}_p[q]$ to *failed*. Notice that processor p has heard about processor q if either $\texttt{Rumors}_p[q]$ or $\texttt{Active}_p[q]$ is not equal to zero.

The purpose of using the array Pending is to facilitate the job of dissemination. If processor p learns that some other processor q is fully informed, that is, it is either a disseminator itself or has been already notified by a disseminator, then it marks this information in $\texttt{Pending}_p[q]$. The array $\texttt{Pending}_p$ is useful even if p is an ordinary processor, because it may relay its local knowledge to a collector by replying to an inquiry. If p is a leader then array $\texttt{Pending}_p$ may be also used to send dissemination messages in a systematic way: it is sufficient to scan $\texttt{Pending}_p$ to find those processors that possibly still have not heard about some processor.

We use the following terminology based on the current contents of the arrays Active and Pending. Processor q is said *to be active according to* p if p has not yet received any information implying that q crashed, which is the same as having a zero in $\texttt{Active}_p[q]$. Processor q is said *to need to be notified by* p if it is active according to p and $\texttt{Pending}_p[q]$ is equal to zero.

Categories of messages. A processor p may send a message to its neighbor in the graph $G(n,t)$, provided that it is is still active according to p, such a message is called a *graph* one. These messages are not sufficient since the communication graph may become disconnected due to failures. Hence other messages are also sent, to cover all the processors in a systematic way, they are called *range* ones. To define how this part of communication works, the processors are partitioned into w *chunks*. The *chunk number* w is equal to $\lceil n/\Delta \rceil$ if $m = n$, otherwise $w = m$. In any case $w \leq m$. Chunk $C(i)$, for $1 \leq i \leq w$, is defined to consist of processors q such that $q \equiv i \bmod w$. Chunks are of balanced size, each of Δ or $\Delta - 1$ elements for $m = n$, and either $\lfloor n/m \rfloor$ or $\lfloor n/m \rfloor + 1$ processors otherwise. Either the equality $C(i) = D(i)$ holds for $m < n$ or sets $D(i)$ are singletons, and then $C(i)$ are of size Δ or $\Delta - 1$. The local permutations are of the integers in the range $[1..w]$. Each leader p considers the chunks as ordered according to its local permutation π_p, that is, it considers them in the order $\pi_p(1), \pi_p(2), \ldots, \pi_p(w)$.

A collector p sends a range message to the first chunk with a processor about which p has not heard yet, and to all such processors in the chunk simultaneously. Each recipient of such a message sends back a *reply* message. Disseminators send range messages also to subsets of chunks. The target chunk selected by disseminator p is the first one containing a processor that still needs to be notified by p, and the message goes to all the processors in the chunk that share this property. Disseminator-range messages need not be replied to: the sender knows already the rumors of all the processors that are active according to it, and the purpose of the message is to disseminate this knowledge.

Messages of the kinds discussed so far are sent during T rounds of communication, where T is called the termination threshold. After that the processors that have not heard about someone yet send direct inquiries to these processors. Such messages are called *last-resort* ones, and are replied to by the non-faulty recipients in the next step.

Updating local state. A message sent by a processor carries its current local knowledge. More precisely, a message

Receiving Step: Receive messages

Computation Step: Update the local arrays

Sending Step 1: If p is a leader then for each processor q: if p has not heard about processor q yet then send a last-resort message to q

Receiving Step: Receive messages

Computation Step: Update the local arrays

Sending Step 2: For each processor q: if a last-resort message was received from q in the receiving step of this phase then send a reply message to q

Receiving Step: Receive messages

Computation Step: Update the local arrays. If p is a collector then it becomes a disseminator

Sending Step 3: If p is a disseminator then send a range message to every processor q that needs to be notified

Receiving Step: Receive messages

Computation Step: Update the local arrays

Sending Step 4: Pause

Figure 3: Before-ending module of Algorithm_Gossip. Code for processor p.

received from processor p brings the following: the id p, the arrays $\texttt{Rumors}_p$, $\texttt{Active}_p$, and $\texttt{Pending}_p$, and a label to notify the recipient about the character of the message. The label is selected from the following: *graph_message*, *range_from_collector*, *range_from_disseminator*, *this_is_a_reply*, *last_resort*; their meaning is self-explanatory.

A processor p scans a newly received message from q to learn about rumors, failures, and the current status of other processors. It copies each rumor from the received copy of $\texttt{Rumors}_q$ into $\texttt{Rumors}_p$, unless it is already there. It sets $\texttt{Active}_p[r]$ to *failed* if this value is at $\texttt{Active}_q[r]$. It sets $\texttt{Pending}_p[r]$ to *done* if this value is at $\texttt{Pending}_q[r]$. It sets $\texttt{Pending}_p[q]$ to *done* if q is a disseminator and the received message is a range one.

If p is itself a disseminator then it sets $\texttt{Pending}_p[q]$ to *done* immediately after sending a range message to q. If a processor p expects a message to come from processor q, for instance a graph one from a neighbor in the communication graph, or a reply one, and the message does not arrive, then p knows that processor q has failed, and it immediately sets $\texttt{Active}_p[q]$ to *failed*.

The algorithm. Algorithm_Gossip is a loop in which phases are iterated. Each phase consists of three steps: receiving messages, local computation, and multicasting messages. Phases are of two kinds: either *regular,* its pseudocode is in Figure 1, or *ending,* its pseudocode is in Figure 2. For $t < \log n$ an additional *before-ending module* is executed between the regular an ending phases. This module consists of four phases, see Figure 3 for its pseudocode. The algorithm starts with all the processors initializing their local objects.

1. Initialize the local objects
2. Perform the regular phase T times
3. If $t < \log n$ then perform the before-ending module
4. Perform the ending phase three times

Figure 4: Algorithm_Gossip.

A processor p initializes the list $\texttt{Rumors}_p$ to zeroes at all the locations, except for the pth one, which is set equal to $rumor_p$. If processor p is a leader then it starts as a collector. A pseudocode of the algorithm is in Figure 4.

Lemma 1. *For any termination threshold $T > 0$ and any family of local permutations Π, Algorithm_Gossip is a resilient gossiping algorithm.*

4. ANALYSIS AND VARIANTS OF THE GOSSIPING ALGORITHM

We start with a randomized version of Algorithm_Gossip. It does not have a family of permutations as a part of its code, instead each leader p generates a random permutation of the integers in $[1..w]$ at the start of the algorithm. This permutation is denoted also by π_p, since its role is exactly the same as if it were a part of the code. The generation is done uniformly, with respect to all the permutations, and independently, over all the processors. The randomized version of Algorithm_Gossip is called Randomized_Gossip.

Property $\mathcal{R}(m,t)$ motivates the following terminology. A subgraph $K \subseteq G(m)^\ell$ is *compact* if the size of K is at least $(m-t)/7$ and the diameter of K is at most $30 \log m+2$. If K is a subgraph of graph $G(m)^\ell$ and $p \in K$ then by the *neighborhood of node v in K* we mean the subgraph containing each node in K whose distance from p in K is at most $30 \log m + 2$. A node $p \in K$ is said to be *compact in K* if its neighborhood in K has size at least $(m - t)/7$. Notice that each node in a compact subgraph is itself compact.

We partition the computation into consecutive *stages:* one stage comprises $30 \log m + 2$ consecutive phases. One stage is sufficient for all the processors in a compact subgraph to communicate. When we refer to a processor during a stage then we mean a processor that does not fail until the end of this stage.

We consider two conceptual lists R_p and S_p, for each leader p. The list R_p is obtained by ordering all the chunks of processors according to π_p, and then removing each chunk with the property that p has already heard about all the elements in this chunk. The list S_p is obtained by ordering all the chunks of processors according to π_p, and then removing each chunk without elements that processor p still needs to notify. Both the lists R_p and S_p are dynamic, and shrink in the course of an execution. If p is a leader then it becomes a disseminator exactly when the list R_p disappears altogether. If p is a collector then it sends range messages to all the processors in the first chunk of R_p that it still has not heard about. If p is a disseminator then it sends range messages to all the processors in the first chunk of S_p that it still needs to notify.

We use the following notations when referring to the behaviour of the two algorithms Algorithm_Gossip and Randomized_Gossip. The notation $r_i(p)$ means the size of list R_p at the end of the ith stage. Let K be a subgraph of $G(m)^\ell$. The notation $r_i(K)$, or r_i if graph K is indicated by the context, means the size of the set of all the chunks which are on any of the lists R_p at the end of stage i, for $p \in K$. More precisely, $r_i(K)$ denotes the number $|\bigcup_{p\in K} R_p|$, where R_p are as at the end of stage i. Similarly, we define $s_i(p)$ and s_i by considering the list S_p instead of R_p.

LEMMA 2. *Let K be a fixed subgraph of $G(m)^\ell$ of diameter at most $30\log m + 2$ at the end of stage $i+1$ of Randomized_Gossip. If $r_{i-1} > \log w$ then the inequality $r_{i+1} \le \max\{r_{i-1} - 3|K|\log m, (2/3)r_{i-1}\}$ holds and otherwise $r_{i+1} = r_i = 0$, with the probability at least $1 - \exp(-\gamma|K|\log m)$, for some positive constant γ. The same estimates hold for s_i.*

LEMMA 3. *Assume $n - t > \log n$. There is a constant $\alpha > 0$ such that if $T \ge \alpha \log w \log m$ then with the probability at least $1 - \exp(-\Omega((m-t)\log m))$ by phase $\alpha \log w \log m$ of algorithm Randomized_Gossip the gossiping has been completed. Moreover, the communication by this phase is*

$$\mathcal{O}\Big(\Big(\frac{n}{n-t}\Big)^{2\log_\lambda \Delta_0} n \log w \log m\Big) .$$

THEOREM 1. *If $n - t = \Omega(n/\log^a n)$, for a positive constant a, then Algorithm_Gossip can be implemented to run in time $\mathcal{O}(\log^2 t)$ and send $\mathcal{O}(n \log^{2+4.35a} t)$ messages.*

PROOF. We use the probabilistic method, relying on Lemma 3. Let us set $T = \alpha \log w \log m$ in Algorithm_Gossip, where the constant α is as in Lemma 3. Let X denote a random variable equal to the amount of communication performed during an execution of algorithm Randomized_Gossip.

Suppose first that $t \ge \log n$, then also $m > \log n$. The event

$$X = \mathcal{O}\Big(\Big(\frac{n}{n-t}\Big)^{2\log_\lambda \Delta_0} n \log w \log m\Big)$$

holds with the probability at least $1 - \exp(-\beta \cdot (m-t)\log m)$, for some positive constant β, by Lemma 3, as no communication is performed during the ending phases. The event $X = \mathcal{O}(n^2 \cdot \log w \log m)$ holds always, since $\mathcal{O}(n^2)$ point-to-point messages are sent at every step of Algorithm_Gossip. Hence we can bound the expected value of X by:

$$\mathcal{O}\Big(\Big(\frac{n}{n-t}\Big)^{2\log_\lambda \Delta_0} n \log w \log m\Big) ,$$

where we used the estimate $(m-t)\log m = \omega(\log n)$. It follows that there exists a family of permutations Π such that if it is used in the code of Algorithm_Gossip then the amount of its communication is at most $\mathsf{E}[X]$. Now we use the estimate of the number $(1+2\log_\lambda \Delta_0)^{-1}$ given in Fact 2, and the estimates $m, w = \mathcal{O}(t)$ and $\frac{n}{n-t} = \mathcal{O}(\log^a n) = \mathcal{O}(\log^a t)$. We obtain the final bounds: $T = \alpha \log w \log m = \mathcal{O}(\log^2 t)$ and $\mathcal{O}\Big(\Big(\log^a t\Big)^{4.35} n \log^2 t\Big)$ is an estimate on the amount of communication.

The analysis of case $t < \log n$ is similar, it relies on the before-ending module instead of the ending phases. □

1. Initialize local objects
2. Perform the regular phase $\alpha \cdot \log^2 t$ times
3. If $t < \log n$ then perform the before-ending module
4. Perform the ending phase three times

Figure 5: Algorithm Balanced_Gossip. The constant α is as in Lemma 3. If $t \ge t_0$ then $G(n, t_0)$ is the communication graph, where t_0 is given by equation (1).

The communication of Algorithm_Gossip is large for the number of failures t close to n. This is because the communication graph $G(n,t)$ has a large degree for large t. To improve upon the communication for large t we could introduce the following two modifications to the algorithm:

- I: Use some fixed graph $G(n, t_0)$ for communication, for $t \ge t_0$, where t_0 is some threshold value.
- II: Stop performing regular phases at some earlier suitable moment, and finish with some direct-communication pattern instead.

The idea is to balance the communication during the two parts of the algorithm. The threshold t_0 can be estimated as follows. During the ending phases the communication is $\mathcal{O}(n(n-t_0))$, also $m = n$ in this case. Select t_0 so that the contributions to the amount of communication from the regular and ending phases are the same:

$$n(n-t_0) = \Big(\frac{n}{n-t_0}\Big)^{2\log_\lambda \Delta_0} n \log^2 n .$$

Solving for t_0 gives:

$$t_0 = n - n^{1-\frac{1}{1+2\log_\lambda \Delta_0}} \log^{\frac{2}{1+2\log_\lambda \Delta_0}} n . \quad (1)$$

The resulting new algorithm is called Balanced_Gossip. It has regular phases executed $T = \alpha \log^2 t$ times. If $t < t_0$ then graph $G(n,t)$ is used as the communication graph, otherwise graph $G(n, t_0)$ serves this purpose. A pseudocode of algorithm Balanced_Gossip is given in Figure 5. We also consider a randomized version of Balanced_Gossip in which the leaders select their permutations in Π randomly at the start of the algorithm, rather than having them as part of their code. This algorithm is called Randomized_Balanced_Gossip, its useful performance characteristic is given in Lemma 4.

LEMMA 4. *The number of point-to-point messages sent by algorithm Randomized_Balanced_Gossip is*

$$\mathcal{O}\Big(\Big(\frac{n}{n-t}\Big)^{2\log_\lambda \Delta_0} n \log^2 t\Big)$$

provided the inequality $t \le t_0$ holds, and otherwise it is

$$\mathcal{O}\Big(n^{2-\frac{1}{1+2\log_\lambda \Delta_0}} \log^{\frac{2}{1+2\log_\lambda \Delta_0}} n\Big) ,$$

with the probability at least $1 - \exp(-\Omega((m-t)\log m))$.

PROOF. First consider the case $t \leq t_0$. The communication during the regular phases is

$$\mathcal{O}\Big(\Big(\frac{n}{n-t}\Big)^{2\log_\lambda \Delta_0} n \log^2 t\Big) ,$$

by Lemma 3 and the estimates $m, w = \mathcal{O}(t)$. It also follows that with the probability $1 - \exp(-\Omega((m-t)\log m))$ all the processors have heard about each other, hence during the ending phases no communication is needed.

Next consider the case $t > t_0$. If during T steps there have been at most t_0 failures then no communication is needed during the ending phases, with the probability at least $1 - \exp(-\Omega((m-t)\log m))$. The amount of communication during the regular phases can be estimated as in the first case for $t = t_0$, and is equal to $\mathcal{O}\Big(n^{2-\frac{1}{1+2\log_\lambda \Delta_0}} \log^{\frac{2}{1+2\log_\lambda \Delta_0}} n\Big)$. If the number of failures is greater than t_0 then the communication during the first T phases is the same, and during the three ending phases the communication is

$$\mathcal{O}((n-t_0)n) = \mathcal{O}\Big(n^{2-\frac{1}{1+2\log_\lambda \Delta_0}} \log^{\frac{2}{1+2\log_\lambda \Delta_0}} n\Big) ,$$

by the definition of t_0. □

Using the probabilistic method, based on Lemma 4, we obtain the following result:

THEOREM 2. *Algorithm Balanced_Gossip can be implemented in such a way that:*
if $n - t = \Omega(n^{0.77}\log^{0.46} n)$ then its time is $\mathcal{O}(\log^2 t)$ and communication is $\mathcal{O}((\frac{n}{n-t})^{4.35} n \log^2 t)$,
otherwise its time is $\mathcal{O}(\log^2 n)$ and communication is $\mathcal{O}(n^{1.77})$.

5. LOWER BOUND FOR GOSSIPING

Suppose that a t-resilient gossiping algorithm $\mathcal{A}$ has at most $\beta n \log^a n$ messages sent in each execution. Our goal in this section is to show that algorithm $\mathcal{A}$ must perform $\Omega(\log n/(\log(n\log n) - \log t))$ steps.

Let us fix n and $t < n$. For each sequence of input rumors $I = (\mathit{rumor}_1, \ldots, \mathit{rumor}_n)$ we construct a strategy $\mathcal{F}_I$ of a t-bounded adversary such that after $b \log n/(\log(n\log n) - \log t)$ steps of algorithm $\mathcal{A}$ there is a processor p_I which has not heard about some other processor q_I.

We will determine a strategy of an adversary to keep at least $\max\{n-t, n/2\}$ processors nonfaulty. Let n_i denote the number of nonfaulty processors after the step i, and let $x > 0$ be a parameter. By the bound on the message complexity, we have that in *each* step of algorithm $\mathcal{A}$ at most $\beta n \log^a n$ point-to-point messages are sent. Observe that during the $(i+1)$th step of communication at most $n_i/(2x)$ processors receive messages from more than $x \cdot \beta \log^a n$ processors. The adversary's strategy is to fail these processors in step i. The numbers n_i satisfy the inequality $n_i \geq n_{i-1}(1 - 1/(2x))$. It follows, by induction on i, that the inequality $n_i \geq n(1 - 1/(2x))^i$ holds, and that each nonfaulty processor has heard about rumors of at most $(1 + x \cdot \beta \log^a n)^i$ processors by step i.

We are constrained by the following inequalities:

$$n \cdot (1 - 1/(2x))^i \quad \geq \quad \max\{n-t, n/2\} \; ; \qquad (2)$$

$$(1 + x \cdot \beta \log^a n)^i \quad \leq \quad n/2 \; . \qquad (3)$$

There are two cases. If $n - t \leq n/2$ then the following estimates hold: $\frac{i}{2x} = \mathcal{O}(1)$ and $i(\log x + \log\log n) = \mathcal{O}(\log n)$, which imply $i = \mathcal{O}(\log n/\log\log n)$. If $n - t > n/2$ then $1 - i/(2x) \geq 1 - t/n$ and $i \log(x \log n) = \mathcal{O}(\log n)$, hence $i = \mathcal{O}(\frac{\log n}{\log(x \log n)}) = \mathcal{O}(\frac{\log n}{\log((n/t)\cdot \log n)})$. Now one can verify directly by simple algebra that there is a constant b such that $i = b\frac{\log n}{\log(n\log n) - \log t}$ and $x = \log(n\log n) - \log t$ satisfy inequalities (2) and (3). We conclude that after $b\log n/(\log(n\log n) - \log t)$ steps there are at least $n - t$ non-faulty processors and each of them has not heard about some other nonfaulty processor.

Let the notation p_I and q_I, for the input rumors I, denote processors such that p_I has not heard about q_I, and both of them are nonfaulty. We still need to show that processor p_I could not decide correctly on the rumor of processor q_I in spite of the lack of communication. We assume that rumors are from an infinite domain, for instance they can be arbitrary positive integers. Denote by $\mathcal{M}_I$ the *message pattern* defined to be the sequence of sets of links used in consecutive steps of an execution of algorithm $\mathcal{A}$. The numbers of different message patterns $\mathcal{M}_I$ and of failure patterns $\mathcal{F}_I$ are finite, as well as the number of pairs of processors p_I and q_I, but the number of possible initial configurations of rumors I is infinite. It follows, by the pigeonhole principle, that there are two different inputs I_1 and I_2, such that the rumor of q_{I_1} is distinct from the rumor of q_{I_2}, while the following are equal: $q_{I_1} = q_{I_2}$, $\mathcal{F}_{I_2} = \mathcal{F}_{I_2}$ and $\mathcal{M}_{I_1} = \mathcal{M}_{I_2}$. Hence, if processor p_{I_1} terminated by step $b\log n/(\log(n\log n) - \log t)$, then it would have to decide what the rumor of processor q_{I_1} is, and that decision would have to be the same that p_{I_2} made in exactly the same time for the input I_2, which is impossible.

This completes the proof of the following lower bound:

THEOREM 3. *If rumors are from an infinite domain and a t-resilient algorithm generates $\mathcal{O}(n\log^a n)$ messages, for some positive constant a, then it requires $\Omega(\frac{\log n}{\log(n\log n) - \log t})$ steps to complete gossiping.* □

6. CONSENSUS ALGORITHMS

Each processor p has a positive integer, from a range of a costant size, as its input value_p. We want the processors to agree on one such value, it will be the maximum one known by the operational processors.

Our consensus algorithms are embedded in the gossiping algorithm Balanced_Gossip. The notations and operations of this algorithm are used throughout this section with the same meaning. During the first run, the value of rumor of processor p is initialized to its input. The notation $\texttt{rumor}_p$ means now a local variable of processor p, this is to maintain the consistency of notations, it is initialized to value_p.

If $t \leq n^{1-\varepsilon}$ then algorithm GMY generates $\mathcal{O}(n + tn^\varepsilon) = \mathcal{O}(n)$ messages, hence we may concentrate on the case $t > n^{1-\varepsilon}$.

Algorithm_Consensus uses phases of a new kind, they are called *spreading.* A pseudo code for spreading phase is in Figure 6. Messages sent out during such phases are also called *spreading,* each such a message carries the current value of $\texttt{rumor}_p$ of its disseminator p.

A pseudo code of Algorithm_Consensus is in Figure 7. Array $\texttt{Rumors}_p$ is initialized in the beginning of each execution of the gossiping algorithm, but using the current value of the variable $\texttt{rumor}_p$. The value of variable $\texttt{rumor}_p$ is kept equal to the current maximum value either in the array $\texttt{Rumors}_p$ or heard from other processors, unless it is set explicitly

Receiving Step: Receive spreading messages

Computation Step: If a value of some variable `rumor` was received in the previous step, such that it is larger than the current value of $\texttt{rumor}_p$ then set $\texttt{rumor}_p$ to the largest among such values

Sending Step: If the variable $\texttt{rumor}_p$ changed its value in the previous step then send a spreading message to each neighbor, in the communication graph $G(n,t)$, that is active according to p

Figure 6: Spreading phase of Algorithm_Consensus. Code for processor p.

1. Initialize the local objects
2. If $t \leq n^{1-\varepsilon}$ then run GMY and exit
3. Perform $\lceil 7m/(m-t) \rceil$ times:
   a. Execute Balanced_Gossip
   b. If p believes to be in a compact neighborhood then set $\texttt{rumor}_p$ to the maximal value stored in $\texttt{Rumors}_p$ else set $\texttt{rumor}_p$ to value_p
   c. Send spreading messages to neighbors in $G(n,t)$ which are active according to p
   d. Perform the spreading phase m times
   e. Update and send $\texttt{rumor}_p$ to the active neighbors $30 \log m + 2$ times
   f. If p does not believe to be in a compact neighborhood then set $\texttt{rumor}_p$ to value_p
4. Execute Balanced_Gossip
5. Decide on the maximum value in $\texttt{Rumors}_p$

Figure 7: Algorithm_Consensus. Code for processor p.

to the input value_p. The number $\lceil 7m/(m-t) \rceil$ of iterations is at least as large, by property $\mathcal{R}(m,t)$, as the number of connected components in the communication graph that contain any compact neighborhood at any moment of an execution. We say that processor p *believes to be in a compact neighborhood* if it is compact in the subgraph induced by the processors that are still active according to p. Processor p can check this property using its array $\texttt{Active}_p$ and the topology of graph $G(n,t)$.

Lemma 5. *Let G be a connected subgraph of $G(m)^\ell$, after parts* c *and* d *of Step* 3 *at Algorithm_Consensus have been performed. Then all the processors p in G store the same value at* $\texttt{rumor}_p$.

Lemma 6. *Algorithm_Consensus is a t-resilient algorithm solving consensus.*

Proof. The value agreed upon is among the initial input ones because in every step each $\texttt{rumor}_p$ stores some among the initial input values.

It remains to show that all the processors agree upon one common value. For each processor p and any moment of computation, the value at $\texttt{rumor}_p$ is at least as large as value_p. Let k_i be the total number of processors in the connected components in $G(m)^\ell$ that contain a compact processor at the end of part b of the ith execution of Step 3, where $i = 1, \ldots, \lceil 7m/(m-t) \rceil + 1$. We may assume that $k_0 = m$. The inequality

$$k_{\lceil 7m/(m-t) \rceil + 1} \geq (m-t)/7$$

holds by property $\mathcal{R}(m,t)$. It follows, by induction, that after the ith iteration in Step 3 the following invariants hold:

Invariant 1. All the processors p, that believe to be compact and are in the same connected component, have the same value of $\texttt{rumor}_p$.

Invariant 2. This common value is equal to at least the maximum of value_q taken over all the processors that have not failed.

Invariant 3. At the end of part b of iteration $(i+1)$, either:

(A) all the processors p that believe to be compact have the same value $\texttt{rumor}_p$, or

(B) the inequality $k_i \leq k_{i-1} - (m-t)/7$ holds.

By gossiping in Step 4, all the processors receive a common value of processors believing to be compact, and this value, by Invariant 2, is maximal among all the rumors. By property $\mathcal{R}(m,t)$, there is at least one processor believing to be compact at the end of the computation, and all the processors decide the same maximum value in (5). $\square$

Let $T(n,t)$ denote the time and $\mathcal{C}(n,t)$ the amount of communication of Balanced_Gossip. In the next lemma we bound the time and communication complexity of Algorithm_Consensus.

Lemma 7. *Algorithm_Consensus terminates in time $\mathcal{O}((t+T(n,t))\frac{n}{n-t})$ and sends $\mathcal{O}((n \cdot \Delta \cdot \log n + \mathcal{C}(n,t))\frac{n}{n-t})$ messages.*

Proof. If $t \leq n^{1-\varepsilon}$ then the time is $\mathcal{O}(t)$ and the number of point-to-point messages is $\mathcal{O}(n)$ by the performance of algorithm GMY.

Suppose $t > n^{1-\varepsilon}$. Observe that $\frac{m}{m-t} = \mathcal{O}(\frac{n}{n-t})$, for all n and t. Hence there are $\mathcal{O}(\frac{n}{n-t})$ executions of Step 3 in the algorithm, see Figure 7. We estimate the time of one execution of Step 3: part a takes $T(n,t)$ steps, part d takes $\mathcal{O}(t)$ steps, the propagating part e takes $\mathcal{O}(\log m)$ steps, the remaining parts take $\mathcal{O}(1)$ steps. Hence the total duration of Step 3 is

$$\mathcal{O}\Big(\big(t + \log m + T(n,t)\big) \cdot \frac{n}{n-t}\Big) = \mathcal{O}\Big(\big(t + T(n,t)\big) \cdot \frac{n}{n-t}\Big) ,$$

since $\log m = \mathcal{O}(t)$ for $t > n^{1-\varepsilon}$. The gossiping procedure in Step 4 takes $\mathcal{O}(T(n,t))$, which is dominated by the contribution from Step 3.

The number of messages sent during one execution of a spreading phase in parts c or d of Step 3 is at most $\mathcal{O}(n\Delta)$. Each processor changes its rumor at most some constant number of times, equal to the number of initial values. Hence, for each link in the communication graph, only a constant number of point-to-point messages are sent during one execution of the spreading phases in c or d. The total number of links in the communication graph is $\mathcal{O}(n\Delta)$. The amount of communication during e is $\mathcal{O}(n\Delta\log m)$, and during a is $\mathcal{O}(C(n,t))$. We obtain $\mathcal{O}(n\Delta\log m + C(n,t))$ as a bound on the number of point-to-point messages sent during one execution of Step 3, hence the total is $\mathcal{O}((n\Delta\log n + \mathcal{C}(n,t))\frac{n}{n-t})$. The amount of communication during Step 4 is $\mathcal{O}(C(n,t))$ which is dominated by the contribution from Step 3. □

Lemma 7 and Theorem 2 imply the following:

THEOREM 4. *Algorithm_Consensus runs in time $\mathcal{O}(n)$ and sends $\mathcal{O}(n\log^2 n)$ messages against any linearly-bounded adversary.*

Next we consider another consensus algorithm, called Balanced_Consensus, which is a modification of Algorithm_Consensus. To describe it, we change the meaning of a compact neighborhood as follows. A node v *believes to be in a* $\frac{m}{2^i}$*-compact neighborhood* if its neighborhood in the subgraph induced by the processors that are still active according to p has size at least $\frac{m}{7\cdot 2^i}$. Instead of executing $\lceil 7m/(m-t)\rceil$ times Step 3 in Algorithm_Consensus, we iterate 14 times only its parts a-f in the stages $i = 1,\ldots,1+\log\lceil m/(m-t)\rceil$. During stage i, we use the notion of a $\frac{m}{2^{i-1}}$-compact neighborhood instead of a compact neighborhood, and iterate the spreading phase $\frac{m}{2^{i-1}}$ times only, in part d of Step 3.

LEMMA 8. *Algorithm Balanced_Consensus is a t-resilient consensus algorithm that runs in time $\mathcal{O}(t + T(n,t)\log t)$ and with $\mathcal{O}(C(n,t)\log t + n\Delta\log^2 t)$ messages.*

Combining Lemma 8 with Theorem 1 yields the following result:

THEOREM 5. *If $n - t = \Omega(n/\log^a n)$, where a is a positive constant, then there is a t-resilient algorithm solving the consensus problem in time $\mathcal{O}(t)$ and with $\mathcal{O}(n\log^{3+4.35a} n)$ messages.*

7. DISCUSSION

We considered the problem of gossiping and consensus in a synchronous setting with crash failures. We developed algorithms that are efficient with respect to both time and communication. The performance of our solutions depends on the power of adversary, defined in terms of the number of crashes it may incur. Our solutions have performance characteristics close to the optimum ones within polylogarithmic factors, if only the number of processors that remain operational is $\Omega(n/\text{polylog } n)$. The problem of developing solutions of a similar performance, or showing limitations to such solutions, for the unbounded adversary, that may fail all but one processor, remains an open problem.

8. REFERENCES

[1] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts, A theory of competitive analysis of distributed algorithms, in *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1994, pp. 401 - 411.

[2] S. Amdur, S. Weber, and V. Hadzilacos, On the message complexity of binary agreement under crash failures, *Distributed Computing,* 5 (1992) 175 - 186.

[3] R.J. Anderson, and H. Woll, Algorithms for the certified write-all problem, *SIAM J. Computing*, 26 (1997) 1277 - 1283.

[4] J. Aspnes, Lower bounds for distributed coin-flipping and randomized consensus, *J. ACM*, 45 (1998) 415 - 450.

[5] J. Aspnes, Time- and space-efficient randomized consensus, *J. Algorithms*, 14 (1993) 414 - 431.

[6] J. Aspnes, and W. Hurwood, Spreading rumors rapidly despite an adversary, *J. Algorithms*, 26 (1998) 386 - 411.

[7] N.T.J. Bailey, "*The Mathematical Theory of Infectious Diseases and its Applications*", Charles Griffin, London, 1975.

[8] Z. Bar-Joseph, and M. Ben-Or, A tight lower bound for randomized synchronous consensus, in *Proc., 17th ACM Symp. on Principles of Distributed Computing*, 1998, pp. 193 - 199.

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

[10] G. Bracha, unpublished manuscript, Cornell University, 1984.

[11] T.D. Chandra, and S. Toueg, Time and message efficient reliable broadcast, in *Proc., 4th Int. Workshop on Distributed Algorithms,* 1990, Springer LNCS 486, pp. 289 - 303.

[12] B.S. Chlebus, L. Gąsieniec, D.R. Kowalski, and A.A. Shvartsman, Bounding work *and* communication in robust cooperative computation, submitted.

[13] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swineheart, and D. Terry, Epidemic algorithms for replicated database maintenance, in *Proc., 6th ACM Symp. on Principles of Distributed Computing,* 1987, pp. 1 - 12.

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

[15] D. Dolev, and R. Reischuk, Bounds on information exchange for Byzantine agreement, *J. ACM,* 32 (1985) 191 - 204.

[16] C. Dwork, J. Halpern, and O. Waarts, Performing work efficiently in the presence of faults, *SIAM J. on Computing,* 27 (1998) 1457–1491.

[17] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal, Fault tolerance in networks of bounded degree, *SIAM J. on Computing,* 17 (1988) 975 - 988.

[18] M. Fisher, and N. Lynch, A lower time for the time to assure interactive consistency, *Information Processing Letters*, 14 (1982) 183 - 186.

[19] M. Fisher, N. Lynch, and M. Paterson, Impossibility of distributed consensus with one faulty process, *J. ACM*, 32 (1985) 374 - 382.

[20] Z. Galil, A. Mayer, and M. Yung, Resolving message complexity of Byzantine agreement and beyond, in *Proc. 36th IEEE Symp. on Foundations of Computer Science*, 1995, pp. 724 - 733.

[21] J.A. Garay, and Y. Moses, Fully polynomial Byzantine agreement for $n > 3t$ processors in $t+1$ rounds, *SIAM J. Computing*, 27 (1998) 247 - 290.

[22] O. Goldreich, and E. Petrank, The best of both worlds: guaranteeing termination in fast randomized byzantine agreement protocols, *Information Processing Letters*, 36 (1990) 45 - 49.

[23] V. Hadzilacos, and J.Y. Halpern, Message-optimal protocols for Byzantine agreement, *Mathematical Systems Theory*, 26 (1993) 41 - 102.

[24] V. Hadzilacos, and S. Toueg, Fault-tolerant broadcast and related problems, in "Distributed Systems", 2nd ed., S. Mullender (ed.), Eddison-Wesley, 1993, pp. 97 - 145.

[25] M. Harchol-Balter, T. Leighton, and D. Lewin, Resource discovery in distributed networks, in *Proc., 18th ACM Symp. on Principles of Distributed Computing*, 1999, pp. 229 - 238.

[26] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking, Randomized rumor spreading, in *Proc., 41st IEEE Symp. on Foundations of Computer Science*, 2000, pp. 565 - 574.

[27] D. Kempe, J. Kleinberg, and A. Demers, Spatial gossip and resource location protocols, in *Proc., 33rd ACM Symp. on Theory of Computing*, 2001, pp. 163 - 172.

[28] L. Lamport, and M.J. Fischer, Byzantine generals and transaction commit protocols, manuscript, 1982.

[29] L. Lamport, R. Shostak, and M. Pease, The Byzantine generals problem, *ACM Transactions on Programming Languages and Systems,* 4 (1982) 382 - 401.

[30] A. Lubotzky, R. Phillips, and P. Sarnak, Ramanujan graphs, *Combinatorica*, 8 (1988) 261 - 277.

[31] G.A. Margulis, Explicit group-theoretical constructions of combinatorial schemes and their applications to the design of expanders and concentrators, *Problems of Information Transmission*, 24 (1988) 39 - 46.

[32] M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, *J. ACM*, 27 (1980) 228 - 234.

[33] A. Pelc, Fault-tolerant broadcasting and gossiping in communication networks, *Networks*, 28 (1996) 43 - 156.

[34] M.O. Rabin, Randomized byzantine generals, in *Proc., 24th IEEE Symp. on Foundations of Computer Science,* 1983, pp. 403 - 409.

[35] M. Saks, N. Shavit, and H. Woll, Optimal time randomized consensus - making resilient algorithms fast in practice, in *Proc. 2nd SIAM–ACM Symp. on Discrete Algorithms*, 1991, pp. 351 - 362.

[36] E. Upfal, Tolerating a linear number of faults in networks of bounded degree, *Information and Computation*, 115 (1994) 312 - 320.

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