Warsaw University
Faculty of Mathematics, Informatics and Mechanics

Michał Strojnowski

Fault-tolerant distributed algorithms

PhD Thesis

Supervisor
dr hab. Damian Niwinski
Faculty of Mathematics, Informatics and Mechanics
Warsaw University

November 2008

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

.....	.....
date	*Michał Strojnowski*

Supervisor's declaration:
the dissertation is ready to be reviewed

.....	.....
date	*dr hab. Damian Niwinski*

Abstract

The following thesis contains several results regarding communication complexity of two basic distributed problems: *gossiping* and *consensus*. Gossiping is a problem of exchanging initial knowledge between all processes. Consensus is a problem of making a common decision by all processes.

The thesis consist of two parts. First part is devoted to the gossiping problem, analyzed in three different fault-prone settings. For all three settings, asymptotically optimal solutions are presented, together with proofs of corresponding lower bounds. These solutions are created on the basis of one common communication framework, introduced in the thesis.

In the second part, the same framework is further extended to create time and message efficient fault-tolerant consensus algorithms. Two algorithms are presented: a deterministic one and a quantum one. In both cases the bit complexity of these algorithms beats best previously known that works in the same time.

The studied model of distributed system is a synchronous message passing system: a system with global clock and point-to-point communication. All units are prone to failures, and therefore protocols can not rely on correctness of any single unit. In some cases there is a bound on a fraction of processor that may fail, which means that in case of more failures safety of the system is not guaranteed. In other cases, correctness is guaranteed as long as at least one processor works properly.

Three types of faults are analyzed: *crash*, *omission*, and *Byzantine* failure. Crash means that processor completely stops its activity at some point. Omission failure means that processor fails to send or receive some messages. Byzantine failure encompasses any malicious behavior of the processor.

The main technique used in this thesis is based on transformation of distributed communication problems to graph problems. Every algorithm is created by definition of underlying communication patterns in a form of graphs with specified properties. Then existence of such graphs is proven using probabilistic methods.

The results are evaluated in terms of execution time, total number of messages used (message complexity) and total length of these messages (bit complexity).

Keywords: gossiping, consensus, distributed algorithms, fault tolerance, adaptive algorithms, processor failures, quantum algorithms

ACM Computing Classification System: B.8.1, C.2.4, G.2.1,

Contents

Chapter 1

Introduction

The topic of this thesis, fault tolerance of distributed algorithms, overlaps with many fields of contemporary computer science. There are many models and notions used to describe distributed systems, useful in different applications (see [7],[61]). Therefore in order to avoid ambiguity, we start with devoting some space to clarify the area that we are interested in. This is the main purpose of this chapter.

Undoubtedly, robustness and stability are desired properties of each computer system. Algorithms by themselves may be flawless, and their implementations may be well prepared for any possible input data. But real world systems often face unpredictable changes in their environment, and sometimes in their physical hardware. Searching for a way of dealing with these errors leads to many interesting fields of research: channel capacity, error correction codes, database consistency, data redundancy and more.

Distributed systems are particularly interesting from this point of view, because there exist many fault types specific only to them. Since distributed system consists of many independent units, it is often unrealistic to require correct behavior of every unit. In fact, it is often expected that system should be able to withstand failure of a substantial part of them. Such circumstances do not need to be caused by real failures in hardware or software. For example, if we perform massive distributed computation in the idle time of a set of PC's connected to the Internet, each time some PC is given any other task, it is removed from our computation. From the point of view of our model, we have to consider such event as a "failure" of this unit.

In many settings, the easiest and cheapest way of dealing with failures is to add some "invulnerable" stations that will oversee the system, assigning work to normal stations and dealing with their failures. The main disadvantage of this solution is, however, creation of a bottleneck. Scalability of such system is bounded by performance of these special stations. And if we start to increase their number, we are back in the first problem, of dealing with a failure of any one of them.

In this thesis we will be considering only fully scalable solutions, with fully homogeneous structure, and not relying on proper behaviour of any of its parts — that is, where all units are treated uniformly and failure of any part does not brake the computation.

When failure occur at some point of an execution, inevitably some information obtained up to this point is lost. It is therefore crucial to maintain redundancy and exchange data in some intermediate steps. If we want to be able to withstand any failures, this task is often expensive, either in time or in number of messages that have to be exchanged. Problems considered in this thesis are crucial to create fast and message-efficient communication when system is severely damaged. The intent of algorithms presented here is to provide effective primitives that may be used as building blocks for more complex protocols.

The structure of the thesis is the following. We start with an overview of distributed systems, which will allow us to define precisely which model we are interested in. Next we formalize our model, using a logical approach. After this introduction, we move to the main results. Chapter 2 is devoted to the gossiping problem, analyzed in several failure models. We present upper and lover bounds for each model, obtaining a strict hierarchy of communication cost incurred by failures of processors. In Chapter 3 we use framework introduced in Chapter 2 to obtain two consensus algorithms: a deterministic one and a quantum one. We end each chapter with a brief discussion of related open problems.

All results contained in this thesis were obtained together with Darek Kowalski during recent years. As always, it is not easy to determine the origin of the ideas behind the solutions, and in most cases both our input was crucial to the final success. Getting into details, I was the author of the first version of gossip against omission failures and consensus against crash failures algorithms presented here, Darek was the author of first version of gossip against crash failures. All these algorithms were then refined by us together. As for the lower bounds, I was the author of lower bound for gossip against omission failures, Darek was the author of lower bounds for gossip against crashes and Byzantine failures. Finally, the quantum consensus algorithm is mine. The results of Section 2 were published in [55], remaining results await publication.

1.1 Overview of distributed systems

Importance of distributed systems is of constant increase in modern computer science. Since upgrading performance of a single processor becomes more and more expensive with each next generation, natural trend is to turn into increasing the number of cooperating units. This, however, leads to many issues of synchronization of their work, sharing the data, parallelization of algorithms,

load balancing, dealing with changes in the system and so on. Efficient solution to these problems (like *Distributed Hash Tables* [11, 77], Bloom filters [16], Paxos consensus algorithms [56, 57], and many more) have stimulated a technological shift in recent years.

One of the standing features of distributed system theory is that there is no single canonical model of such system. In contrast to classical theory of computation, where all computing devices are usually modeled as equivalent to Turing Machine, there exist many nonequivalent models of distributed systems. This is mainly motivated by real-world application. Commercial systems utilize radio networks, cable networks, multiprocessor systems, and each of them in many variants. Although it is possible in some cases to simulate one by another, usually it is too expensive. Instead, different algorithms are designed for any of these models.

For this reason, distributed systems spawned a great number of results, theorems and algorithmic techniques. Describing the whole landscape of them is far beyond the scope of this thesis. Here we present only a brief overview of models existing in literature, in order to give a reader an outlook on the place of presented results in the whole theory.

In general, each distributed system model may be identified by fixing a set of parameters. These parameters are:

Structure of the system:

- Homogeneous
  In homogeneous system, all units are of the same type. All units are prone to the same type of failures and usually no assumptions about robustness of any units is made. Safety of such system is defined often in terms of fraction of units that may fail without violating the algorithm outcome.

- Heterogeneous
  Heterogeneous systems may contain different kinds of units with various abilities and efficiency. This is extremely useful for practical purposes if we can introduce some "central" unit or units, assumed to be flawless. Such technique is used, e.g., in massive Internet-based computation like SETI@home [72]. Such systems are particulary hard to describe by a theoretical model, as they stand in-between distributed and classic model of computation.

Synchronicity of the system:

- Asynchronous systems
  In asynchronous system, every unit works according to its internal clock,

which may arbitrarily differ from the clocks of other units. It is known that some problems (eg. deterministic consensus with at least one failure [37]) are unsolvable in this model.

- Synchronous systems
  In synchronous system, we assume existence of some *global clock*, which produces time reference accessible to all units. Such feature allows very efficient communication between units, in particular deriving information from lack of communication or from the time frame of a message. Synchronous systems also often assume that all machines start simultaneously. In other models i.e. initially only one machine is working, and other machines *wake up* after receiving first message.

- Partially synchronous systems
  One can find many intermediate models between the two presented above. We may assume e.g. that there is an upper bound on the time between sending and receiving any message, but it is not known to machines (if it was, the system could be considered synchronous). In other setting, we may assume that such bound only starts to hold in some unknown time T [33]. Other models assume an upper bound on clocks speed quotients, or introduce some types of synchronizers [9] or other failure detectors [12, 26, 30, 31].

Communication structure of the system:

- Fully connected systems
  In fully connected system, every unit may communicate directly with any other.

- Network systems
  In network system only some pairs of processors may communicate directly, others have to route their messages through the network. This is modeled by defining a graph of connection between units. Existence and efficiency of many algorithms in such system depends on the properties of this graph - e.g. connectivity, degree, diameter and so on.

- Non-graph systems
  More exotic models of radio network utilize also physical properties of radio waves. If we take into account that a message is received when the signal-to-noise ratio is high enough then interaction between simultaneous transmission becomes much more complicated, and graph model is no longer valid.

Initial knowledge of every unit:

- Whole system known
  In such setting, every unit knows from the beginning the identity of all units and the network structure. In particular units have unique identifiers.

- Neighborhood known
  Every unit knows only identity of its neighbors, *a priori* not knowing anything about units that it can not communicate directly with. If a system is fully connected, this setting is identical to the previous one.

- Ad-hoc setting
  In ad-hoc system, no initial information about structure is given to the units. It is a reasonable model for mobile dynamic networks, where units may emerge and vanish from the system at any time.

System evolution in time:

- Static system
  Structure of such system does not change during time that we are interested in. This often greatly simplifies the analysis, and is reasonable especially when we analyze short-time algorithms and protocols.

- Dynamical system
  Dynamical systems are much more complicated to analyze, and are further divided into many subsequent models. We may permit adding new units and/or removing the old ones (permanently or temporarily), appearance and disappearance of communication links in various patterns and so on.

Communication primitives:

- Message passing
  Units communicate by sending messages to each other. This is the most basic model of distributed communication. Depending on the setting, we may assume that receiver can always identify the sender of each message, or that sender may claim false identity.

- Shared memory
  Units communicate by placing some data in a memory accessible to every or to some set of units. This model is motivated especially by multiprocessor integrated system. Policy of access to shared memory further divides it into several models.

- Radio transmission
  Broadcast to all neighbors is allowed. Usually it is assumed that in any given moment only one message may be received, and if more than one neighbors broadcast, considered unit does not receive any message (but sometimes it can detect a collision). Radio networks introduce several new communication issues, but some techniques from message-passing systems are used also in this context [22, 23, 40]. Special case of radio networks are geographical networks, where units are positioned in some space and links are defined by distances between them (e.g. unit disk graph [24]).

Properties listed above are independent to a great degree, and one can obtain many reasonable models taking different combinations. Many of such models are not only interesting from the theoretical point of view, but have also some practical analogs in the real world. Others may be considered as simplified versions of more complex ones, and may be used for deeper analysis of desired problems. For example, the whole Internet may be treated as partially synchronous, heterogeneous, dynamical network where every node knows only its neighbors and only message passing is allowed. Multiprocessor systems may be, on the other hand, treated as a synchronous, static, fully connected system with shared memory. Smart dust and sensor networks may be modeled by asynchronous, dynamical ad-hoc radio networks [4].

The rest of this thesis in devoted to homogeneous, synchronous, fully connected system with known structure and message-passing communication, which is a canonical message passing system used in literature [7, 63]. One may think of this system as an simplified model of peer-to-peer network, where every unit knows all peers. Although real peer-to-peer network is usually heterogeneous, only partially synchronous and has some topology, many algorithms designed for a simpler model may be relatively easily implemented on it. We now define our model in more formal way.

1.2 Message passing systems

Message passing system consists of any number of units, that communicate by sending point-to-point messages. Since we consider a synchronous system, execution is performed in rounds. In a single round, every unit may perform any finite computation, send any number of messages and receive any number of messages. This models real world systems, in which time between sending and receiving a message is several orders of magnitude longer than the time required for internal processor operation. Because of that, in many cases execution time of distributed algorithm is determined by the time spent on waiting for messages. In this model we measure only this time, with a single message travel time being a unit of measure. In real world systems, this time

usually varies, therefore to obtain meaningful result we must assume some upper bound on it. One may think of the time complexity considered here as a communication "depth", rather than a number of operations to perform.

In this section we formalize this setting. Most notions and definitions are intuitively understandable, but in order to avoid ambiguity we define them in a formal way, using terms from computation theory. One may find more refined formal definition of such model, called *Hybrid I/O automata*, in [62].

1.2.1 Units, rounds and communication

A distributed message passing system S_n is a set of n units along with a set of communication interfaces. Each unit is a deterministic Turing Machine with many tapes. In this thesis we call it TM, machine, unit, process or a node, using these terms as synonyms. Each communication interface represents a directed edge between two units. It consist of two buffers: *send buffer* and *receive buffer*. Any pair of units (A, B) may be connected with an interface. In such case, send buffer is a write-only unidirectional tape connected to machine A, and receive buffer is a read-only unidirectional tape connected to machine B.

In general, the set of communication interfaces may create any directed graph on the set of machines. In this thesis we consider only fully connected case, where there are $n(n-1)$ communication interfaces. Every machine has $n-1$ send buffers and $n-1$ receive buffers.

Each machine executes its own program, which may (and usually does) include sending and receiving messages. Each machine starts knowing number n and its own unique identifier $i \in \{1, \ldots n\}$, both written on its working tape. For simplicity, we assume that the code (also called *protocol* or *algorithm*) P is the same for all n machines. Since every machine starts with different i, in fact behavior of machines may differ. Also, behavior of machines may be changed by *failures*, occurring in the system.

We are interested only in uniform algorithms, that is, algorithms with a single predefined code for all n. Since the set of tapes depends on n, this code translates to different transition tables for every n.

Each machine may have some additional input written initially on its working tape. All send and receive buffers are initially empty. All machines start simultaneously. Protocol ends when all correct machines (but not necessarily the faulty ones) enter terminal state. We will formalize later when such protocol *solves* a given problem. It will always depend on data stored on the working tapes at the beginning and at the end of the computation.

To formalize the notion of a round, we introduce two synchronizing states for each machine: *StartRound* and *FinishRound*. Each machine that enters *FinishRound* state, stays idle in this state until rest of machines do the same. This assumption makes our model synchronous. In a real system, we may

obtain similar result by requiring each machine to wait some predefined time after each round, to ensure that each other machine finishes its own tasks (unless it is faulty). There exist many more sophisticated ways to make real system synchronous, but we shall not discuss them here, noting only that our model does not require any unrealistic assumptions.

When all correct machines enter *FinishRound* state, the following happens:

- Content of every *receive buffer* is replaced with the content of *send buffer* of the same interface. We define this operation as atomic, regardless of the number of symbols in send buffer.
- Content of every *send buffer* is cleared.
- Heads on send and receive buffers are placed on the beginning.
- Every machine enters *StartRound* state.

This simulates sending and receiving messages in a synchronous manner. Every nonempty content of every buffer is called a *message*. With this formalization, *sending a message* to process **p** means simply to write its content on a proper send buffer. *Receiving a message* means to copy content of receive buffer to working tape.

We note here that this formalization differs in one detail from the automaton model (as, e.g., in [61]). Namely, machine may control order of messages sent in every round. This is especially important in case of crash failures, which may occur during sending messages (see the next subsection). In the automaton model it is assumed that any subset of messages may be delivered in such case, while in our formalization only some sets may be possible, depending on the protocol. It means that our model in slightly stronger, and potentially may allow more efficient algorithms. However, it will be easy to see that it is not the case for the problems considered here, because all of our algorithms as well as all proofs of lower bounds work in both models.

For consistency, we assume that initial state of each machine is *StartRound*, and there is no transition leading to this state in other way than described above. This guarantees that all correct machines will enter this state in the same moment and the same number of times.

1.2.2 Failures

We now formalize our model of failures. This formalization is equivalent to the classic model of processor failures, as defined in literature (see e.g., [7, 63, 70]).

We say that a machine is *faulty* when it executes protocol different from the implemented one. We remind here that the code for every machine is the same. For now, we assume that initially the set of machines is divided into

two parts: the set of correct ones and the set of faulty ones. This division is not known to the machines, and they may only infer failures of their peers by analyzing messages received from them. Assumption that the set of faulty machines is predefined is sufficient as long as we consider only deterministic algorithms (which are the main topic of this thesis). In the last section, when we consider quantum algorithms, we discuss extension of this model.

Since faulty machines may never end their computations, we say that protocol finishes when all correct machines enter terminal state. We call a protocol *t-resilient*, if in any setting with up to t faulty machines, and for any possible behavior of faulty machines allowed by a given failure type, the correct machines terminate and terminal state meets requirements defined in the problem.

We define three types of failures, and thus three corresponding resiliency types. Let P be an initial protocol of machine M.

Crash failures We call a computation C of machine M a *crash failure*, if it meets the following requirements:

- There exists a point X in computation C, up to which M executes only transitions allowed by P (thus this is a valid computation of code P).
- After point X, machine enters final state.

Observe that this may also be a behavior of a correct machine (when it enters final state in a legal way). This is an important intuition for our failure model. Faulty machines *may behave* correctly. The definition of t-resiliency says only that protocol must withstand *any* allowed behavior of the faulty machines.

Omission failures We call a computation C of machine M an *omission failure*, if it uses only the following moves:

- transitions of M,
- *empty write* move - instead of writing message to send buffer, machine may ignore it, leaving send buffer empty,
- *empty read* move - instead of reading message from receive buffer, machine may ignore its content, acting like if the receive buffer would be empty.

Such notion allows machine to ignore any data on any input tape and drop any data intended to be written on any output tape. It is crucial that messages in this model are not altered, only ignored.

Byzantine failures *Byzantine failure* encompasses all possible behaviors of machine M. In particular, such machine may incorrectly read any value from any input tape, or write any message on the output tape. Even more, behavior of these machines need not to be predefined in a form of transition table. They may, for example, read and write on communication tapes of each other (but only those of the faulty ones). In short, set of faulty machines may send in every round any combination of messages — even not a computable one. t-Byzantine-resilient protocol must work correctly for such behavior of any set of t machines.

It is worth mentioning an existence of a slight modification to this model, which might be introduced to prevent faulty machines from forging messages created by the correct ones. It is called an *authenticated Byzantine* model [29]. In this model, system is equipped with some public key infrastructure. Each piece of data may be *signed* by its creator, so that any other machine can verify which machine is its original source. It is assumed that such signatures are unforgeable. Then if only signed pieces of data are accepted, no faulty machine can claim falsely that some correct machine sent some information. This sometimes allows us to use omission-resilient algorithms as a Byzantine-resilient ones [29].

Note that crash failures are most benign kind of failures, while Byzantine faults are most severe among considered ones. Indeed, a crashed processor could be seen as one suffering an omission failure that performs *empty write* on every tape starting from some given point. On the other hand, Byzantine and authenticated Byzantine processor may simply omit some sent/received messages, thus mimicking an omission fault.

1.2.3 Complexity measures

Execution finishes correclty, if at some point all correct machines enter terminal state, and additional requirements defined by the problems are met. At this point we may calculate a cost of this execution. We use the following costs definitions:

Communication complexity or **message complexity** is a number of messages sent during the execution. Formally, we count how many times a nonempty content of a send buffer of some correct machine is copied to a receive buffer of any machine. It is important that messages sent to faulty machines are counted, but not messages sent *by* faulty machines.

Bit complexity is a total number of bits of the messages considered in the previous case.

Time complexity is a number of times when each correct machine entered *StartRound* state. Since all correct machines enters this state in the same moment, this number is identical for all of them.

All costs are represented as the asymptotic functions of the number of processors n, the resiliency threshold t (maximum number of failures that protocols handle) and the number of failures f (actual number of faulty processors during the execution). In gossiping problem we use also additional parameter r, being the total length of initial data of all processors. This parameter affects only the total length of all messages, and it is important only in the bit complexity.

1.3 Expanders

Our approach to construct efficient distributed algorithms is based on definition of special communication patterns, expressed in a form of graphs with required properties.

All graphs used in this thesis belong to family of expanders [51]. Such graphs are in general hard to construct deterministically, but usually quite easy to obtain if edges are placed randomly. There is no single definition of expander graph. In different contexts, several slightly different expansion measures are used, to determine if a given graph is an expander.

Most useful property for our purposes is *vertex expansion*. We say that a given undirected graph $G = (V, E)$ has a vertex expansion α for sets of size β, if

$$\forall_{S \subset V} |S| < \beta|V| \Rightarrow |\Gamma(S)| \geq (1 + \alpha)|S|,$$

where $|\Gamma(S)|$ is a set of neighbors of a set S, that is, a set of vertices with at least one neighbor in S. For a bipartite graph often the definition is reformulated to include only subsets of one side of the graph: we say that bipartite undirected graph $G = (A, B, E)$ with $E \subseteq A \times B$ has expansion rate α for sets of size β on side A, when

$$\forall_{S \subset A} |S| < \beta|A| \Rightarrow |\Gamma(S)| \geq (1 + \alpha)|S|.$$

It means that if we take any sufficiently small set of vertices, the number of its neighbors will be greater by some multiplicative factor. In distributed system this may be used in an straightforward manner: if we start to distribute some information from one node, in each step sending it to all neighbors, in logarithmic number of steps this information will reach at least βn vertices. If our expander is sparse (has $O(n)$ edges), such process will require sending only $O(n)$ messages.

In this thesis, we modify those definitions to obtain some other expander-like graph features. The common property of all these features is that they

are met with high probability by random graphs. In most cases there are no direct translation from these properties to expansion rate, but the analogy is strong enough to consider them as a expander graphs.

Essentially, expanders provide basic communication scheme similar to spanning trees, which is very useful in efficient distribution of information. Unlike the spanning trees, expander may however be very fault-resilient, preserving their good properties even when some nodes are removed. Existence of such graphs, and quantitative analysis of their properties is one of the main foundation of results in this thesis.

1.3.1 Spectral expansion

In this thesis we also show how to construct some of graphs that we use. Our technique is based on creating a graph with large *spectral gap*. If G is a regular graph with degree d, its adjacency matrix has eigenvalues which are real numbers $\lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_k$, and $\lambda_0 = d$. The difference $d - \lambda_1$ is called a spectral gap, and is closely related to vertex expansion of the graph. If we consider graphs of expansion α on sets of size $\frac{n}{2}$ then we have a dependency:

$$\frac{1}{2}(d - \lambda_1) \leq \alpha \leq \sqrt{2d(d - \lambda_1)}$$

There are known efficient algorithms to construct graph with high value $d - \lambda_1$. We construct our graphs from Ramanujan graphs [60].

1.4 Related work

Message-passing was one of the first model of distributed computing, and there is a broad literature of classical results regarding it, as well as many recent research ([7, 18, 38, 61, 63, 66] and many more). Crash failures were also among the first considered [7, 37, 61]. Probably the best known result about this setting is so-called FLP theorem, stating that in fully asynchronous system there exists no deterministic 1-crash-resilient consensus algorithm [37].

Omission model of failures was introduced by Hadzilacos in [45]. Subsequently it was divided into send-omission and general-omission models [67]. Neiger and Toueg showed in [63] that each crash-resilient algorithm may be transformed into omission-resilient, but with big communication overhead $O(nt)$. Results in this thesis improve their results, giving simulation with overhead $O(n + tf)$, where f is the actual number of faults in the execution. In both cases simulation works in constant time.

Byzantine failure model takes its name form *Byzantine Generals Problem* [58, 65]. In this problem, several generals from Byzantium have to decide unanimously whether to attack enemy or to withdraw. There are some traitors

among them, which try to create some type of diversion. This problem was extensively studied in distributed system, as it encompass many possible incursion to the real systems. It is also a foundation of consensus problem, which is deeper analyzed in the third chapter of this thesis.

Dolev and Reischuk in [29] considered Byzantine agreement with signatures added to the messages, and showed lower bound $O(n + t^2)$ on the number of signatures required to solve gossip and consensus problems in this setting.

Expander techniques were previously used to create fault-tolerant communication in message-passing system [19, 28, 41, 73], networks in general (for references see e.g., [17]) and shared memory [21].

Graphs with fault-tolerant properties were studied in the context of networking algorithms [73], implementing fault-tolerant data structures [8], cooperative distributed algorithms [41] and others. t-fault-tolerant graphs were recently re-introduced by Chlebus et al. [21] in the context of collect algorithms in shared-memory distributed environments. These authors showed how to construct such graphs for $t = \Omega(n)$ number of failures.

In all these papers graph of at least logarithmic diameter were used. We extend these ideas to graphs of constant diameter, and combine several different expander-like properties in our protocols.

Chapter 2

Gossiping

Gossiping problem is defined as follows: Initially each processor has unique piece of information called a *rumor*. The goal is to distribute rumors in the system, such that every processor knows all the rumors.

It was firstly introduced in early '70 as a combinatorial problem, initially examined without any failures [10, 48, 75]. Later it became abstraction of a simple all-to-all message exchange, used in many distributed systems. It represents any situation when processors need to compare gathered data, performed work, information about network etc. Classical algorithms that use this subroutine are linear problem solving and discrete Fourier transform [50].

One of the examples that use gossiping is *do-all problem* [42]. There is a set of independent tasks to complete, and set of unreliable stations performing them. By periodically exchanging information about tasks completed so far, all stations may keep record on tasks that are still to be done. Each machine finishes only when it knows that all task are already finished. In this case, each rumor could be a set of tasks completed by a given processor.

Gossiping is closely related to broadcast problem, in which one processor wants to distribute its knowledge in the whole system. Here all processors must perform such task, and one may think of gossip as a simultaneous broadcasts of all processors.

We start this chapter with proper formalization of the gossip problem, together with an outline of the previous and new solutions. Then in Section 2.2 we define common framework that we use to solve this problem in all cases. Next sections contain solutions to gossiping: Section 2.3 with crash failures, Section 2.4 with omission failures and section 2.5 with Byzantine failures. Each section starts with a proof of lower bound, which is followed by algorithm that meets it. Additional remarks and special cases are discussed at the ends of sections. Section 2.6 contains brief discussion about remaining open problems in this area.

2.1 Problem setting

Definition of the problem in our formalization is the following: In the initial state, each machine M_i has some data (*rumor*) r_i, written on its working tape. All rumors have finite size, and total length r of the rumors is a parameter of the problem.

The goal is to distribute these rumors in the system, such that every unit knows all the rumors. Formally, we say that protocol P *successfully solves* gossip, if for any set $\{r_1, r_2, \ldots, r_n\}$ of initial rumors, in accepting state every unit has somewhere on its working tape a sequence:

$Rumors : r_1 \# r_2 \# \ldots \# r_n$,

and symbol $Rumors$: is written exactly once on this tape (to prevent multiple guesses). In a faulty setting, a set F of faulty machines is introduced. We say that protocol P successfully solves gossip, when every correct machine ($M \in A \backslash F$) in accepting state knows rumors of all correct machines. That is, it has on its tape a sequence:

$Rumors : x_{M,1} \# x_{M,2} \# \ldots \# x_{M,n}$

such that for every $i \in A \backslash F$, $x_{M,i} = r_i$. Again symbol $Rumors$: is written exactly once. It is often useful to require that for every faulty machine, correct machines should either receive its rumor or information that it is faulty. However, such requirement is impossible to meet for Byzantine failures - such machine may, for example, send faulty rumor to all machines and apart from that behave correctly. Therefore, we do not place any requirements on rumors of faulty machines. Luckily, all protocols presented in this thesis meet this additional requirement in case of crash and omission failures.

Observe that correct machines may know different sequences of rumors in the accepting state, because for any faulty machine some may receive its rumor and some don't. If we require these sequences to be identical, we would in fact need to solve general consensus problem: in this case, the set of known rumors would be a piece of information that all correct processors must agree on. Such problem is considered in Chapter 3 in a simplified form (when this piece of information contains only one bit).

We say that gossip protocol P is t-resilient, if for every set of faulty machines F such that $|F| \leq t$, P successfully solves gossip. Depending on type of failures, we obtain definitions of t-crash-resiliency, t-omission-resiliency and t-Byzantine-resiliency.

2.1.1 Overview of the previous results

There is a vast body of literature concerning gossiping problem. We now present important techniques that were used to solve it in fully-connected message passing system.

We start by pointing out that obvious fault-tolerant solution to gossiping problem is for every processor to send its rumor to every other processor. In the second round, every processor copies obtained rumors to its working tape and enters terminal state. This solution works for any number of faults, and is time-optimal. The main disadvantage is that every processor sends $n-1$ messages, which yields total message complexity $\Theta(n^2)$. On the other hand, we can easily see that bit complexity of this algorithm is optimal. In any execution without failures, each node must receive all rumors. If total length of rumors is r bits, lower bound on bit complexity of gossip problem is therefore $r(n-1)$ bits - which is exactly bit complexity of this naive algorithm. This solution shows that it is easy to obtain gossip algorithm optimal in terms of time and bit complexity. The only interesting task is to obtain low message complexity. All algorithms presented below concentrate on this.

In case when there are no failures, there exists a trivial optimal solution [48, 75]. We pick one leader, to whom all processors send their rumors. Then the leader sends back combined rumors to all processors. This solutions requires three rounds: in the first round messages are sent, in the second round leader combines received rumors into a single message and sends it to every processor, and in the third round every processor copies obtained message to its working tape and enters terminal state. Number of messages is $2n-2$.

This solution no longer works when we introduce failures, because leader may become faulty and do not send any message. To circumvent this problem, one may introduce many leaders. Dolev and Reischuk [29] used this idea to create an algorithm with $2t+1$ leaders as a building blocks for Byzantine agreement. In case of Byzantine failures, rumors may be forged by faulty leaders, and processors must validate them somehow. In this algorithm, processor that receive many versions of the same rumor, chooses the most common one. Hence the number of leaders $2t+1$, which guarantees that the correct rumor occur at least $t+1$ times, which is more than any faulty rumor may occur.

Dolev and Reischuk have also shown that in case of Byzantine failures their algorithm is optimal, because any t-Byzantine-resilient algorithm requires $\Omega(nt)$ messages.

The same solution works also for crash and omission failures, but is far from being the best possible. Of course in this case we need only $t+1$ leaders, because there are no forged rumors an we need only to make sure that each rumor will be retransmitted. But the number of messages might be significantly lowered.

For crash failures, there were several papers published, that concentrated on lowering the number of messages, at the expense of prolonging the algorithm. Chlebus and Kowalski [19] developed an algorithm resilient to $t < cn$ crashes, for a positive constant $c < 1$. Their algorithm was based on expander communication graphs and had time complexity $O(\log^2 n)$ and message com-

plexity $O(n \text{ polylog } n)$. The reason for bounding the number of failures was to rely on the fact that constant fraction of processors is correct during the execution.

That result was extended for any possible $t < n$ by Georgiou et al [41] who presented a gossip algorithm working in time $O(\log^2 n)$ and with message complexity $O(n^{1+\varepsilon})$, for any constant $0 < \varepsilon < 1$. Later, Chlebus and Kowalski [20] developed a solution tolerating up to $n - 1$ crashes in time $O(\log^3 n)$ and with $O(n \log^4 n)$ message complexity. Their algorithm works by iterating a simpler algorithm, where each iteration ends correctly if only a constant number of processors fail. Logarithmic number of such iterations is sufficient to solve gossip for any number of failures. We use similar technique in this thesis, extending it so that only a constant number of iterations is required.

Note that complexity of crash resilient gossip is generated mostly by the fact that crashes occur dynamically. That is, each round new processors may crash. In model with static crashes, when each processor is either crashed from the beginning or correct till the end, it is possible to solve gossip in time $O(\log n)$ and with $O(n)$ messages. Such algorithm was developed by Diks and Pelc [28].

Omission failure model was much less examined in this context. Neiger and Toueg in [63] showed that each crash-resilient protocol may be transformed into an omission-resilient, by adding a gossiping in each round of communication. They used previously mentioned algorithm with leaders, yielding communication overhead $\Theta(nt)$ each round. Results in this thesis improves their theorem, lowering the overhead to $O(n+tf)$ messages for each gossiping, when f defines the number of failures that occur during the execution.

2.1.2 Related problems

Gossiping is one of the basic building blocks of networking algorithms. It is used to maintain routing tables in peer-to-peer networks [76], ensure proper data replication in distributed databases [27], gathering information about failures [68], recognition of neighborhood [49], finding nearest resource [54] and many more problems. A reader interested in finding many more applications may use a survey of Pelc [66]. A book [52] presents some aspects of fault-tolerant solutions of the gossip problem in general networks.

In the literature, problems similar to gossiping are considered also in other settings [40, 53]. In shared memory model of computation, problem called *collect*, was introduced by Saks et al. [69]. Gathering and spreading information is studied also from slightly different perspective [1, 3, 6, 21].

2.1.3 New results

In this chapter we present asymptotically optimal solution to gossip problem for crash, omission, authenticated Byzantine, and Byzantine failures. We define a common framework to create protocols, based on special classes of graphs. In each case we accompany algorithm with corresponding lower bound to show its optimality. In this section we shortly overview these results, giving a reader intuition about how costly the gossip problem really is.

A straightforward lower bound for the message complexity is $\Omega(n)$, since each processor must send its rumor at least once. As we have seen, in the model without failures, this lower bound is matched by the algorithm with one leader.

In this chapter we prove the following results: If we consider only constant-time algorithms (that is, working in the same number of rounds for any n and t) then the situation is as follows.

Optimal t-crash-resilient algorithm for gossiping uses $\Theta(n+fn^{\varepsilon})$ messages. Here $\varepsilon > 0$ is an arbitrary preselected constant. It means that, for any ε, there exists a constant-time algorithm with communication complexity $O(n+fn^{\varepsilon})$, and for any constant k, there exists ε such that any algorithm solving gossiping it time k has asymptotic complexity $\Omega(n+fn^{\varepsilon})$.

Optimal t-omission-resilient algorithm uses $\Theta(n+tf)$ messages.

Optimal t-Byzantine resilient algorithm uses $\Theta(nt)$ messages (this is result by Dolev and Reischuk [29]).

The costs for authenticated Byzantine failures are the same as for omissions failures.

In terms of cost incurred by each faulty processor we may rewrite it in the following way: every crash failure costs additional $\Theta(n^{\varepsilon})$ messages, every omission/authenticated Byzantine failure cost additional $\Theta(t)$ messages, and every (even potential) Byzantine failure costs $\Theta(n)$ messages. Table 2.1 summarizes the results.

Failure type	Message complexity	Cost per failure
crash	$\Theta(n+fn^{\varepsilon})$	$\Theta(n^{\varepsilon})$
omission	$\Theta(n+tf)$	$\Theta(t)$
authenticated Byzantine	$\Theta(n+tf)$	$\Theta(t)$
Byzantine	$\Theta(nt)$ [29]	$\Theta(n)$

Table 2.1: Message complexity incurred by different kinds of failures in constant-time gossip. n - number of processors, t - maximal tolerated number of failures, f - actual number of failures.

There are several immediate consequences of these results. First, it is easy to see that it is possible to create protocol that solves gossip in a constant time

for any n using $O(n)$ messages, if the number of crashes is bounded by $n^{1-\varepsilon}$, for some $\varepsilon > 0$ (execution time will depend on this ε). In case of omission and authenticated Byzantine failures the same is possible only for number of failures bounded by $\sqrt{n}$. For Byzantine failures is is possible only for constant number of failures.

Second, only for crash failures it is possible to choose a cost of the execution arbitrarily close to $O(n)$, by changing value ε. In all remaining cases, minimal cost is predetermined. For example, in often considered case $t = n - 1$, lower bound in omission case is $\Omega(n(f + 1))$, and in Byzantine case is $\Omega(n^2)$, while in crash case is only $O(n^{1+\varepsilon})$. It means that only in case of crash failures there may be non-constant trade-off between time and message complexity.

Third consequence is that only in Byzantine case there are no adaptive algorithms. Adaptive algorithm is an algorithm whose cost depends on the actual number of failures that happened during the execution. It is crucial for real systems, where failures happen rarely, and it is desired to have resilient solution which is cheap in most cases.

These results complete analysis for constant time gossip. In this thesis we also consider gossip algorithms working in non-constant time. As we shall see in the proofs of lower bounds, such algorithms are no better in case of omission and Byzantine failures. In both cases lower bounds on message complexity are the same as in constant time, and message optimal solutions are possible in constant time. This is not the case for crash failures, where there is a tradeoff between time and message complexity. We present a parameterized algorithm that shows this trade-off. For any preselected s, it solves gossip in time $O(\log_s^3 n)$ and using $O(ns^2 \log_s^2 n)$ messages. On the lower bound size, we present a proof that any protocol working in time c must use at least $\Omega(n + fn^{1/c})$ messages.

2.2 Framework

We start by defining a common framework for our algorithms. It includes a specification of local memory, types of messages, update procedures, communication patterns and subroutines.

The most complex parts of this framework are communication patterns, which are defined as graphs with expander-like properties. We use classical probabilistic methods to prove their existence. We do not present an efficient deterministic construction of all of these graphs, nor efficient deterministic verification. We use their existence to prove existence of our deterministic protocols. Also, we define required properties of these graphs is a form that allows deterministic verification in an exponential time. It means that as long as the number of computational steps is unimportant, every processor may obtain all required graphs in the first round, simply enumerating all graphs

of size n and taking the first one with the required properties for each class. This substantiates uniformity of presented algorithms. In a real system, we expect that those graphs would be precomputed and given to processors as an input (in fact, it is enough for each processor to know only its neighbors in these graphs). Probabilistic proofs of the existence of these graphs show also that one can easily transform these protocols into efficient probabilistic Monte Carlo algorithms with arbitrarily high probability of success.

2.2.1 Local memory, messages and updates

Local memory Each processor $\mathbf{p}$ stores the following structures:

- $Rumors_{\mathbf{p}}[1 \dots n]$: array of rumors known to processor $\mathbf{p}$. Each field $Rumors_{\mathbf{p}}[\mathbf{q}]$ contains rumor of processor q or one of the special values: *faulty* or *unknown*. Initially $Rumors_{\mathbf{p}}[\mathbf{p}] = rumor_{\mathbf{p}}$, and remaining fields contain values *unknown*.

- $Active_{\mathbf{p}}[1 \dots n]$: array of activities of processors. Each field $Active_{\mathbf{p}}[\mathbf{q}]$ contains value *unknown* (initial) or *active* (meaning of this value is described in algorithms).

- $Informed_{\mathbf{p}}[1 \dots n]$: array of statuses of processors in the gossip. Each field $Informed_{\mathbf{p}}[\mathbf{q}]$ contains value *unknown* (initial) or *informed* (which means that processor $\mathbf{q}$ has collected all required rumors)

- $BrokenLinks_{\mathbf{p}}$: list of known pairs of processors that failed to communicate. This list is used only in the gossip against omission failures. Each entry informs that at least one processor from this pair is faulty.

All above arrays and lists are gradually filled during the execution. In particular, array $Rumors_{\mathbf{p}}$ is filled with rumors of correct processors and values *faulty* for faulty processors. We say that a processor $\mathbf{p}$ is *informed* if it has no fields *unknown* in array $Rumors_{\mathbf{p}}$. If all non-faulty processors are informed, gossip is solved.

Additionally, all processors know the same family of predefined graphs, specified in the algorithm.

Messages and memory updates For simplicity, all messages used in our algorithms have the same format. Every message contains three of four local memory structures described above: $Rumors_{\mathbf{p}}$, $Informed_{\mathbf{p}}$ and $BrokenLinks_{\mathbf{p}}$. We emphasize here that whenever algorithm requires sending any message, either this is a request, reply or some other message, it always has this content. Type of every message is determined by round number, since in every round messages of only one type will be sent.

It is clear that every message in algorithm against crash or byzantine failure have size $\Theta(n)$, and in algorithm against omission failures may have size $\Theta(n+tf)$ (because tf is upper bound on the size of $BrokenLinks_{\mathbf{p}}$ list). Since in the ideal case all these structures should be filled with the same data, all processors will be updating their local knowledge with all data obtained in messages. The procedure is as follows. At the beginning of every round, every processor reads received messages and overwrites all its local values *unknown* whenever there is known corresponding value in any of these messages. Additionally, processor update list *BrokenLinks*, creating a list that contains all pairs known so far, together with all pairs received in messages.

If there are more than one possible update (which may happen e.g. when one message contains rumor of some processor, while other contain information that it is faulty), we only assume that some of them is made (in this particular case, probably more reasonable would be to chose actual rumor, but this does not influence correctness of the algorithm).

2.2.2 Communication graphs

Before describing the main subroutines of the algorithms, we need to define three classes of graphs: *distributors*, *communicators* and *adaptive communicators*. These graphs will be used as parameters in communication subroutines, out of which we build our protocols. Roughly speaking, in a subroutine we identify processors with nodes of used graph, and processors send messages only to their neighbors in these graphs. Therefore graph properties correspond to message complexity and other characteristics of a subroutine.

The motivation for introducing new graph structures is to apply them in gossip algorithm in the following way: communicators provide good connectivity in a given set, assuring that after removing some part of it, there is still a large subgraph with small diameter. Distributors provide failure-proof connection between two sets, allowing the smaller set to gather from the larger set or distribute data in the larger set.

Distributors are graphs which are similar to loss-less unbalanced expanders, while communicators are related to constant-diameter concentrators [17]. However both relations are not strict and there are some differences between the corresponding structures which cause that the graphs used in this paper can not be directly replaced by others in our fault-tolerant applications.

All graphs presented below are undirected, and nodes/processors usually communicate both ways along the adjacent edges (unless stated otherwise). For a graph $G = (V, E)$ and a subset of nodes $B \subseteq V$, we denote by $N_G(B)$ a set of all neighbors of B in graph G (this definition also includes those nodes in set B that have a neighbor in B).

Distributors Distributor is a bipartite unbalanced expander graph. Let $G = (W, L, E)$ be a bipartite graph with W and L being disjoint sets of nodes, and $E \subseteq W \times L$ being a set of edges. We say that G is an (n, x, Δ)-*distributor* iff it satisfies three following properties:

(a) $|W| = n$, $|L| = \frac{2n}{x}$;

(b) maximum degree of node in set W is at most Δ;

(c) for every $f < \frac{2n}{x^2}$, every set $Y \subseteq W$ of size $\frac{4f}{\Delta}$ has more than f neighbors.

Set W is called a set of *workers* and set L is called a set of *leaders*.

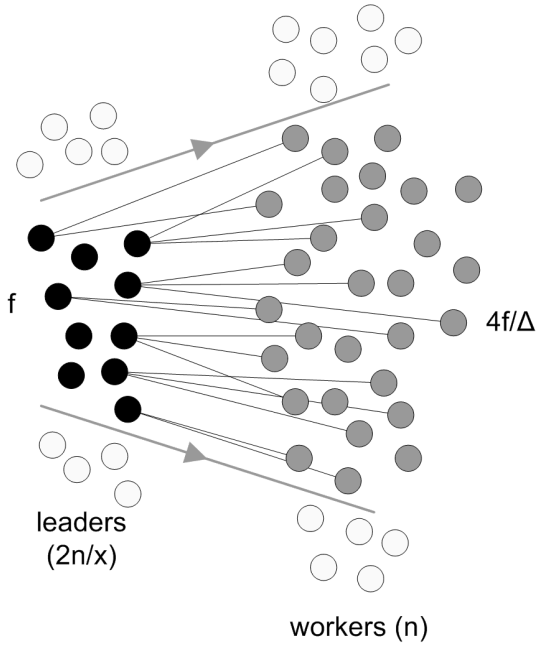

Figure 2.1: Distributor: Each sufficiently big set of workers (grey) has big set of neighboring leaders (black) (only some edges presented)

Theorem 2.1 *There exists an* (n, x, Δ)*-distributor, for any* n, $\Delta \geq 4$ *and* $x \geq 8$.

Proof: We proceed with probabilistic method. We create a random graph and show that it has all required properties with nonzero probability.

Consider a bipartite random graph $G = (W, L, E)$, where $|W| = n$, $|L| = \frac{2n}{x}$, and set of edges E is defined as follows: for every node $v \in W$, we select a set of Δ neighbors from set L uniformly and independently from the other nodes in W. Since graph G has properties (a) and (b) by construction, it remains to show that the probability that graph G satisfies condition (c) is positive.

Fix $f < \frac{2n}{x^2}$, set $X \subseteq L$ of size f and set $Y \subseteq W$ of size $\frac{4f}{\Delta}$. We compute the probability that $N(Y) \subseteq X$. This probability is equal to the product, over all nodes $v \in Y$, of the probabilities that $N(v) \subseteq X$.

$$\mathbf{Pr}\left(N(Y) \subseteq X\right) = \left(\frac{f}{|L|}\right)^{\Delta|Y|} = \left(\frac{fx}{2n}\right)^{4f}.$$

We now compute the probability that for a given f, there exists in graph G subsets $X \subseteq L$ of size f and $Y \subseteq W$ of size $\frac{4f}{d}$, such that $N(Y) \subseteq X$. To simplify the computation we observe that $\frac{4f}{\Delta} \leq f < \frac{n}{2}$, so $\binom{n}{4f/\Delta} \leq \binom{n}{f}$.

$$\begin{aligned}
\binom{|L|}{|X|}\binom{|W|}{|Y|}\mathbf{Pr}\left(N(Y) \subseteq X\right) &\leq \binom{2n/x}{f}\binom{n}{f}\left(\frac{fx}{2n}\right)^{4f} \\
&\leq \left(\frac{2ne}{fx}\right)^{f}\left(\frac{ne}{f}\right)^{f}\left(\frac{fx}{2n}\right)^{4f} \\
&\leq \left(\frac{e^2 f^2 x^3}{8n^2}\right)^{f} \leq \left(\frac{e^2}{2x}\right)^{f} \leq 2^{-f}
\end{aligned}$$

The probability that G does not satisfy condition (c) is no greater than probability that there exists f such that we can find appropriate sets X and Y.

$$\sum_{1 \leq f \leq \frac{2n}{x^2}} 2^{-f} < 1 \ .$$

The probabilistic argument completes the proof of the existence of the required distributor.

□

Communicators Communicator is a graph in which every large subset of nodes contains a large subgraph with small diameter. Graph $G = (L, E)$ with n nodes and degree Δ is an (n, x, Δ)-*communicator*, iff

> For each $B \subseteq L$ of size $m \geq \frac{6nx}{\Delta}$ there exists $C \subseteq B$ of size more than $\frac{m}{2}$, such that the subgraph of G induced by set C has diameter at most $2\log_x n$.

Theorem 2.2 *There exists an (n, x, Δ)-communicator for any n, Δ and $x \geq 2\log n$.*

Proof: Let us define a random graph $G = (V, E)$ of n vertices and set of edges E chosen as follows: for every node $v \in V$, we choose Δ times a random neighbor, uniformly selecting from the set of all nodes, and connect it to v by an edge. In this process some edges may be created twice, but for simplicity

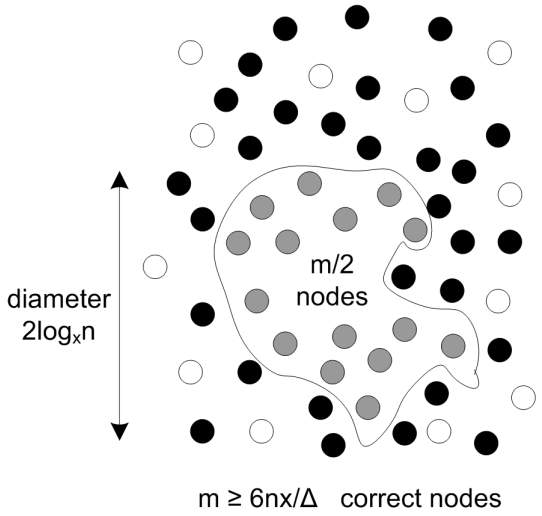

Figure 2.2: Communicator: After removing any set of nodes (black), there remains a big set with small diameter (grey) (edges not displayed)

we just accept this. We will be referring to v-chosen edge as the edge that was obtained by selection for node v (thus, some edges may be chosen for two nodes).

We shall prove that this graph has the required property with positive probability.

Fix a and a set $B \subseteq V$ of $b = \frac{6nx}{\Delta}$ nodes. Let $Y \subseteq B$ be any set of a nodes. There were $a\Delta$ random selections of neighbors of nodes in this set. We consider them as a $a\Delta$ ordered random trials, introducing any order on them. We say that the trial is *successful* if a node from B is selected for the first time in considered trials, or the set of already selected neighbors in B has size more than $\frac{b}{2}$. Every trial is therefore successful with probability at least $\frac{b}{2n} \geq \frac{3x}{\Delta}$. Expected number of successes is at least $a\Delta \cdot \frac{3x}{\Delta} = 3ax$. From the Chernoff bound, we get that the probability that there are less than ax successes is at most

$$\mathbf{Pr}\left(S < ax\right) = \mathbf{Pr}\left(S < (1 - \frac{2}{3})3ax\right) < \left(\frac{e^{-\frac{2}{3}}}{(\frac{1}{3})^{\frac{1}{3}}}\right)^{3ax} = z^{ax}$$

Where $z = \left(\frac{e^{-\frac{2}{3}}}{(\frac{1}{3})^{\frac{1}{3}}}\right)^3 < \frac{1}{5}$

From now on we consider only the graph induced in G by vertices in B. Using the fact above, for every $v \in B$ we may prove the following facts about its neighborhood in this graph:

(1) The probability that there are at least x nodes in his 1-st neighborhood is at least $1 - z^x$.

(2) The probability that there are at least $x(x+1)$ nodes of distance at most 2 from node v is at least $(1-z^x)(1-z^{x^2})$.

...

($\log_x n$) The probability that there are more than $\frac{b}{2}$ nodes of distance at most $\log_x n$ from node v is at least $(1-z^x)(1-z^{x^2})\dots(1-z^{x^{\log_x n}}) \geq 1 - \sum_{i=1}^{\log_x n} z^{x^i} \geq 1 - 2z^x$.

Let us say that a node v is B-expanding, if v has more than $\frac{b}{2}$ nodes of distance at most $\log_x n$.

It follows that a node v is not B-expanding with probability at most $2z^x$. Note that set of B-expanding nodes has diameter at most $2\log_x b$. Indeed, each of them has more than $\frac{b}{2}$ nodes in distance $\log_x n$, so each pair has at least one common node in this distance.

To determine a probability that G is not a communicator, it is sufficient to count the probability that there exists a set B of size $b \geq \frac{6nx}{\Delta}$ such that in the graph induced by B we can find at least $\frac{b}{2}$ not B-expanding nodes. Recalling that $x \geq 2\log n$, this probability is bounded by:

$$\begin{aligned}
\sum_{\frac{6nx}{\Delta} \leq b \leq n} \binom{n}{b}\binom{b}{\frac{b}{2}}(2z^x)^{b/2} &\leq \sum_{\frac{6nx}{\Delta} \leq b \leq n} \left(\frac{ne}{b}\right)^b \cdot 2^b \cdot n^{-b} \\
&\leq \sum_{\frac{6nx}{\Delta} \leq b \leq n} (2e/b)^b \\
&\leq \sum_{\frac{6nx}{\Delta} \leq b \leq n} 2^{-b} < 1
\end{aligned}$$

The probabilistic argument completes the proof of existence of the required communicator.

□

Communicators posses one additional property that we need in our analysis:

Theorem 2.3 *In (n, x, Δ)-communicator, each two distinct sets of at least $\frac{3nx}{\Delta}$ nodes are connected with an edge.*

Proof: Let A and B be two disjoint sets of size at least $\frac{3nx}{\Delta}$ nodes each. If we take a set $C \subseteq A \cup B$, with equal number $k \geq \frac{3nx}{\Delta}$ nodes from both sets, it will have a subset of diameter at most $2\log_x n$ and more than k nodes, thus containing nodes from both sets.

□

Adaptive communicators Adaptive communicator is a graph in which removal of a small set of nodes never detaches a bigger set from the main connected part. Formally, a graph $G = (L, E)$ of n nodes and degree d is an (n, s, d)-*adaptive-communicator*, iff:

> For each $f \leq \frac{n}{s}$ and $B \subseteq L$ of size $n - f$ there exists $C \subseteq B$ of size at least $n - 2f$, such that the subgraph of G induced by C has diameter at most $2 \log_d n$.

Theorem 2.4 *There exists an (n, x, Δ)-adaptive-communicator for any n, $x \geq 6$ and $\Delta \geq 36$.*

Proof: We prove this by showing that Ramanujan graphs [60] of sufficiently large degree are adaptive communicators.

It is known that every graph $G = (V, E)$ of n nodes and spectral expansion $h = \frac{\lambda_1}{\lambda_0}$ has for every $\alpha < 1$ node expansion $\frac{1}{(1-\alpha)h^2+\alpha}$ on sets of size αn: that is, for every $S \subset V$ of size αn, $|N(S)| \geq \frac{|S|}{(1-\alpha)h^2+\alpha}$.
Particularly, Ramanujan graphs of degree c have spectral expansion $h \leq \frac{2\sqrt{c-1}}{c}$. Simplifying this to $h^2 \leq \frac{4}{c}$, we can rewrite it as two properties:

1. $\forall_{|S| \leq \frac{8n}{c}} |N(S)| \geq \frac{|S|}{\frac{4}{c}+\frac{8}{c}} \geq \frac{c|S|}{12}$.
2. $\forall_{|S| > \frac{8n}{c}} |N(S)| \geq \frac{|S|}{(1-\frac{8}{c})\frac{4}{c}+\frac{|S|}{n}} > \frac{2n}{3}$.

Let G be a Ramanujan graph of n nodes and degree $c = 24\Delta$. Let $f < \frac{n}{6}$ and $G_B = (B, E)$ be a graph obtained by removing f nodes from G. We now consider neighborhood only in this graph.

Lemma 2.5 *Every set $Z \subseteq B$ of f nodes contains a node that has more than $\frac{n}{2}$ neighbors in distance $\log_\Delta n$.*

Proof: Denote the i-th neighborhood of a set X in G_B by $N^i(X)$. We define a sequence of sets Z_i in a following way: $Z_0 = Z$, and for every i, Z_{i+1} is the set of nodes $v \in Z_i$ which $(i+1)$-st neighborhood is either at least Δ times bigger than i-th, or has size greater than $\frac{n}{3\Delta}$. That is:

$$Z_{i+1} = \{v \in Z_i : |N^{i+1}(v)| \geq min\{\Delta |N^i(v)|, \frac{n}{3\Delta} + 1\}\}$$

Now, let $D_i = Z_i - Z_{i+1}$.
By induction, we can show that $N^i(Z_i)$ has always at least f nodes:
First note that $N^0(Z_0) = Z$.
Now we prove the induction step:
(a) If $|N^i(Z_i)| \geq f$, then from the expanding property of G (and from the fact that we removed f nodes from G): $|N^{i+1}(Z_i)| \geq \min\{\frac{2}{3}n, 2\Delta f\} - f$.

(b) If $|N^i(D_i)| > \frac{f}{\Delta}$, then from the fact that $\forall_{v \in D_i} N^i(v) \leq \frac{n}{3\Delta}$, and $f \leq \frac{n}{6}$, we could find $D \subseteq D_i$ such that $\frac{n}{3\Delta} \geq |N^i(D)| > \frac{f}{\Delta}$. Then, $|N^{i+1}(D)| < \Delta|N^i(D)| < 2\Delta|N^i(D)| - f$, which contradicts the node expansion property of G. We conclude that $|N^i(D_i)| \leq \frac{f}{\Delta}$, and $|N^{i+1}(D_i)| \leq \Delta \cdot \frac{f}{\Delta} = f$.

(c) Combining this two bounds:

$$|N^{i+1}(Z_{i+1})| \geq |N^{i+1}(Z_i)| - |N^{i+1}(D_i)| \geq \min\{\frac{2}{3}n, 2\Delta f\} - 2f \geq f$$

as long as $2\Delta \geq 3$.

This means that $\forall_i Z_i \neq \emptyset$. Particulary, $Z_{\log_\Delta n - 1}$ contains only nodes with more than $\frac{n}{3\Delta}$ neighbors in distance $\log_\Delta n - 1$. Each such node has more than $\frac{2}{3}n - f \geq \frac{n}{2}$ neighbors in distance $\log_\Delta n$. This completes the proof.

□

Since $|B| = n - f$, it follows from the lemma, that there exists a set $C \subseteq B$ of $n - 2f$ nodes, in which every node has this property. Diameter of graph induced by C will therefore is at most $2\log_\Delta n$, because every two nodes will have at least one common node in distance $\log_\Delta n$. This completes the proof of the Theorem.

□

2.2.3 Subroutines

We use the graphs defined above in two simple subroutines. Each of them takes as one of the parameters a graph, being one of the graphs predefined in the algorithm. All correct processors are executing these subroutines simultaneously, using the same graphs as a parameters. Consequently, they also end the subroutines simultaneously.

DistributedRequest$(graph, param)$:
This subroutine is usually executed by leaders in some *distributor graph*, in some cases it takes as a parameter *communicator graph*. The aim of this subroutine is to collectively request a set of processors for response. Code for processor $\mathbf{p}$ is the following:

- Round 1: Send messages to some neighbors in $graph$, depending on the $param$:
  - To all, if $param = all$
  - To those $\mathbf{q}$ for which $Rumors_{\mathbf{p}}[\mathbf{q}] = unknown$, if $param = unknown$
  - To those $\mathbf{q}$ for which $Informed_{\mathbf{p}}[\mathbf{q}] = unknown$, if $param = uninformed$

- Round 2: Reply for every received message.
- Round 3: For each $\mathbf{q}$ that did not answer for a message sent in Round 1, set $Rumor_{\mathbf{p}}[\mathbf{q}] = faulty$.

Mixing$(graph, rounds)$:
This subroutine takes as a parameter some *communication graph*. The purpose of this subroutine is to exchange the knowledge in some set of processors. Parameter *rounds* defines number r of rounds of this subroutine. Parameter *graph* defines graph G that is used to communicate. In every round, every uninformed processor that belongs to G, sends messages to all its neighbors in G.

Note that this subroutine is in fact a small gossip. Observe that if there is any set S of correct processors that induces in G a subgraph of diameter at most r, all these processors exchange their initial knowledge. In our protocol we use the fact that such set S always exists. The algorithm does not identify this set, but its correctness is usually proven using the existence of some such set.

Waiting When only a part of the system is performing some subroutine, other processor will *wait* in their rounds. This is indicated by instruction "wait for x rounds" in our algorithms. Because other processor may not know if their neighbors are waiting, we need to allow them to reply for request send to them. On the other hand, we do not want waiting processors to send unnecessary messages. This is realized in the following way. In first round of waiting, processor does not send any messages. In subsequent rounds, it replies for each message it received. Our algorithms have the property that no processor sends any message in the round just before waiting starts, therefore two waiting processors will not be exchanging any messages.

2.3 Crash failures

We are now ready to present our first protocol, that solves gossip in a presence of crash failures.

Crash failures are relatively easy to handle, comparing with the other classes of faults. The reason for this is the following: whenever some processor $\mathbf{p}$ expects a message from $\mathbf{q}$ and does not receive it, it immediately knows that $\mathbf{q}$ crashed. Moreover, $\mathbf{p}$ may distribute this knowledge to other processors, and other processors do not need to validate it. This allows us to propagate the knowledge about faults together with gossip data, and efficiently remove crashed processors from the protocol. With sufficiently fault-tolerant communication, additional cost for each faulty processor is minimized. In

fact, the only lower bound in this case comes from the speed of information dissemination in the system. We start by proving this bound.

2.3.1 Lower bound

Theorem 2.6 *Any crash-resilient gossip algorithm terminating in c rounds, must use $\Omega(n + fn^{\frac{1}{c+1}})$ messages, when f failures occur.*

Proof: Consider a t-crash-resilient gossip algorithm $\mathbf{P}$, for some $t < n$. Let $\varepsilon = \frac{1}{c+1}$. The lower bound $\Omega(n)$ is obvious from the fact that in the execution without failures each processor must send and receive at least one message. We show the lower bound $\Omega(fn^{\varepsilon})$, by proving that for some sequence of crashes, $\mathbf{P}$ sends more than $\frac{fn^{\varepsilon}}{2}$ messages. We consider only nontrivial case when $n^{\varepsilon} > 2$ and $f \leq n - 2$.

Assume the contrary, that for each sequence of $f \leq t$ crashes, algorithm $\mathbf{P}$ uses at most $M = \frac{fn^{\varepsilon}}{2}$ messages. We show a sequence of crashes that leads to incorrect outcome.

The sequence starts as follows: in each round, each processor that has received at least n^{ε} messages till this round, crashes. By assumption, the number of such processors in the whole execution is at most $\frac{M}{n^{\varepsilon}} = \frac{f}{2}$. In the last round some additional processors are crashed, so that the total number of crashes is f.

Now we need to pick up these last round crashes in a way to obtain incorrect outcome. We first show that no processor knows rumors of all other processors by the end of round c. We observe that each non-crashed processor receives less than n^{ε} messages. Let k_i denote the size of biggest set of rumors known to a single processor in round i. It is also the upper bound on number of rumors sent in a single message in round i. Because in first round every processor knows only one rumor, we have $k_1 = 1$ and $k_{i+1} \leq n^{\varepsilon}k_i + 1$. By induction, $k_j \leq \sum_{i=0}^{j-1} n^{i\varepsilon}$ for $j \leq c$. Consequently, $k_c \leq \sum_{i=0}^{c-1} n^{i\varepsilon} < n^{1-\varepsilon} < \frac{n}{2}$. It means that no processor knows more than $\frac{n}{2}$ rumors at the end of round c. Let us take any non-crashed processor $\mathbf{p}$ at the end of round c. Until now at most $\frac{f}{2}$ processors were crashed. $\frac{n}{2} + \frac{f}{2} \leq n - 1$, so there exists a processor $\mathbf{q}$ which is non-crashed and its rumor is not known to $\mathbf{p}$. If we crash any set of processors that does not contain $\mathbf{p}$ nor $\mathbf{q}$, the execution ends incorrectly. This violates crash-resiliency of this algorithm.

□

Corollary 1 *Every crash-resilient gossip algorithm terminating in constant time, sends $\Omega(n + fn^{\varepsilon})$ messages, for some constant $\varepsilon > 0$.*

2.3.2 Algorithm

We now show how to solve the gossip task in the presence of crash failures in constant number of rounds and with $O(n + fn^{\varepsilon})$ message complexity, for any given $\varepsilon > 0$. Our algorithm GosCrash consists of two parts. Part I (see Figure 2.3) attempts to solve the gossip using a small predefined set of leaders L, of size $2n^{1-\varepsilon/6}$. Leaders exchange knowledge among themselves and communicate collectively with the rest of the system. Part I uses $O(n)$ messages, independently of the number of failures. If majority of the leaders is correct (in particular if $f < n^{1-\varepsilon/3}$), after this part the gossip is completed and all processors know all the rumors.

Part I iterates *Phases*: first the *gathering phases* to gather the rumors in the set L then the *informing phases* to spread the gathered information to all correct processors. The processes of gathering and informing are symmetric to each other, in the sense that only the direction of the flow differs. In each *Phase*, leaders exchange data gathered so far, and then perform collective communicating task. In gathering phases this task is to gather rumors unknown so far to the leaders. In informing phases this task is to send the gathered set of rumors to processors that might not have it yet. Parameter i of a *Phase* determines maximum number of leaders that may communicate with the same processor. In each phase, the number of requests sent to every remaining processor increases, but simultaneously, the number of remaining processors decreases. The algorithm is designed in a way to maintain linear number of messages in each phase.

All correct processors that are uninformed after Part I, perform Part II (see Figure 2.4). During that part the informed processors are waiting (see page 31). The only action that they perform is to send a reply for every message they receive. We note here that receiving a message from an informed processor makes the receiver informed. Unlike Part I, Part II is an adaptive process in which the algorithm tries to gradually change the parameters in the consecutive subroutines. *Phases* in Part II are similar to *Phases* in Part I. The main difference is that slightly different graphs for mixing and requests are used, to handle bigger number of crashes during the execution.

Since the number of correct processors might become small during the execution, algorithm gradually adapts to such case by increasing the number of messages sent in each round by every processor. This is captured by an intermediate procedure *Epoch*. Each *Epoch*(i) is similar to Part I, but uses more messages and successfully solves gossiping when the number of correct processors is at least $\frac{4n}{s^{i-1}}$. Part II costs at most $O(n^{1+\varepsilon/3})$ messages and solves the gossip for any number of failures. Since for $f < n^{1-\varepsilon/3}$ no processor sends any message in Part II (because all are informed after Part I), the total message complexity of this part is always $O(fn^{\varepsilon})$. The details follow.

For a given constant parameter $0 < \varepsilon \leq 1$, let $k = \lceil \frac{6}{\varepsilon} \rceil$ and $s = \lceil n^{\varepsilon/6} \rceil$. Let

L (the set of leaders) be the set of first $\lceil \frac{2n}{s} \rceil$ processors, and let A denote the set of all processors. Let G_{AC} be a predefined $(|L|, 6, s)$-adaptive-communicator on a set L. Let for any $i \leq k$, G_{Di} be a predefined (n, s, s^i)-distributor with set of leaders L and set of workers A. Let for any $i \leq k$, G_{Ci} be a predefined (A, s, s^i)-communicator on all processors.

Note that for simplicity set of workers is a set of all processors, therefore leaders act also as a workers, executing two programs simultaneously. Total number of nodes of the graph is $|L|+|A| = \lceil \frac{2n}{s} \rceil + n = \Theta(n)$, which means that it does not affect asymptotic costs. Also, there is only one format of message, therefore these two programs do not collide (in each communication interface in each round message is send or not, but there is never a need to send two messages simultaneously). Also, sending a message to oneself each processor may simulate without communication interface.

Each processor runs Part I of the algorithm (see Figure 2.3). If after Part I it is not informed then it runs Part II (see Figure 2.4). Otherwise, it waits during the period corresponding to the execution of Part II, and it only replies for all messages received during this period.

2.3.3 Analysis

Correctness of the algorithm comes from the fact that in the first line of *Epoch*$(s, k-1)$ in Part II, uninformed processors send requests to all their unknown neighbors in (n, s, n)-communicator (which is a complete graph), and receive answers. After running this line, all correct processors are informed.

Each *Phase*$(i, j, param)$ lasts for $6k + 3$ rounds, each *Epoch*(i) lasts for $2(k+1-i)(6k+3)$ rounds. Whole algorithm lasts for $k(k+1)(6k+3) = O(k^3)$ rounds. In case when k is a constant, time complexity of the algorithm is a constant. We now analyze the message complexity. The following lemmas describe the progress and the message complexity of the algorithm after Part I and Part II.

Lemma 2.7 *Message complexity of Part I is $O(n)$.*

Proof: Graph G_{AC} has $\lceil \frac{2n}{s} \rceil$ vertices and degree s, therefore it has $O(n)$ edges. Since it is used constant number of times in each Mixing, message complexity of this subroutines is $O(n)$. We focus on the message complexity of DistributedRequests.

It is enough to prove that during each *Phase*(i) at most $\frac{8n}{s^i}$ processors are requested by leaders. Indeed, since left-degree of (n, s, s^i)-distributor G_{Di} is at most s^i, each processor may receive at most s^i requests, and the total number of requests and answers is $\frac{8n}{s^i} \cdot 2s^i = O(n)$.

Now we prove the above-mentioned fact, for gathering phase. Proof for informing phase is analogous. Let $\mathbf{p}$ be any leader that performs DistributedRequest in *Phase*(i), and let $U_{\mathbf{p}}$ be a set of processor whose rumor is unknown

Phase$(i, param)$**:**

1. Run **Mixing**$(G_{AC}, 2k)$.
2. If $\mathbf{p} \in L$:
   - fill array $Active_{\mathbf{p}}$ with values *unknown*
   - compute the number of *unknown* values in array
     - $Rumors_{\mathbf{p}}$, if $param = gathering$
     - $Informed_{\mathbf{p}}$, if $param = informing$
   - if this value is at most $\frac{8n}{s^i}$, set $Active_{\mathbf{p}}[\mathbf{p}] = active$
3. Run **Mixing**$(G_{AC}, 2k)$.
4. If $\mathbf{p} \in L$:
   - if there is more than $\frac{|L|}{2}$ values *active* in array $Active_{\mathbf{p}}$, leave $Active_{\mathbf{p}}[\mathbf{p}] = active$ and set the remaining fields to *unknown*
   - else, fill the whole array $Active_{\mathbf{p}}$ with values *unknown*
5. Run **Mixing**$(G_{AC}, 2k)$.
6. If $Active_{\mathbf{p}}[\mathbf{p}] = active$ and there are more than $|L|/2$ values *active* in $Active_{\mathbf{p}}$, run **DistributedRequest**$(G_{Di}, param)$
   else, wait for 3 rounds.

Algorithm GosCrash**, Part I:**

- For $i := 0$ to k run *Phase*$(i, gathering)$
- For $i := 0$ to k run *Phase*$(i, informing)$

Figure 2.3: Algorithm GosCrash, Part I, code for processor $\mathbf{p}$.

to $\mathbf{p}$ after the first Mixing in this phase. Since $\mathbf{p}$ marked itself *active*, $|U_{\mathbf{p}}| \leq \frac{8n}{s^i}$. We now prove that all requests in this phase are sent only to workers in $U_{\mathbf{p}}$.

Assume that some $\mathbf{p}'$ performs DistributedRequest in *Phase*(i). For any node x, let A_x denote a set of values *active* in array $Active_x$ after the third Mixing in this phase. Both sets $A_{\mathbf{p}}$ and $A_{\mathbf{p}'}$ are larger than $|L|/2$. Thus there exists $\mathbf{q} \in A_{\mathbf{p}} \cap A_{\mathbf{p}'}$. Since before the third Mixing, $\mathbf{q}$ was in distance at most $2k$ from $\mathbf{p}$, in second Mixing all rumors known to $\mathbf{p}$ were transmitted to $\mathbf{q}$, and in third Mixing all were transmitted to $\mathbf{p}'$. It means that $\mathbf{p}'$ knows all rumors outside the $U_{\mathbf{p}}$, which completes the proof.

Phase$(i, j, param)$**:**

1. Run **Mixing**$(G_{Ci}, 4k)$.
2. Analyze neighborhood:
   - Fill array $Active_{\mathbf{p}}$ with values *unknown*.
   - compute the number of *unknown* values in array
     - $Rumors_{\mathbf{p}}$, if $param = gathering$
     - $Informed_{\mathbf{p}}$, if $param = informing$
   - if this value is at most $\frac{3n}{s^{j-2}}$, or if $i = j$, set $Active_{\mathbf{p}}[\mathbf{p}] = active$
3. Run **Mixing**$(G_{Ci}, 2k)$.
4. Analyze neighborhood:
   - if there is more than $\frac{3n}{s^{i-1}}$ values *active* in array $Active_{\mathbf{p}}$, leave $Active_{\mathbf{p}}[\mathbf{p}] = active$ and set the remaining fields to *unknown*
   - else, fill the whole array $Active_{\mathbf{p}}$ with values *unknown*
5. Run **Mixing**$(G_{Ci}, 2k)$.
6. If $Active_{\mathbf{p}}[\mathbf{p}] = active$ and there are more than $\frac{3n}{s^{i-1}}$ values *active* in $Active_{\mathbf{p}}$, run **DistributedRequest**$(G_{Cj}, param)$
   else, wait for 3 rounds.

Epoch(i):

- For $j := i$ to k run *Phase*$(i, j, gathering)$
- For $j := i$ to k run *Phase*$(i, j, informing)$

Algorithm GosCrash**, Part II:**

- For $i := 1$ to k:
  - if $\mathbf{p}$ is informed, wait for $2(k + 1 - i)(6k + 3)$ rounds
  - else, run *Epoch*(i)

Figure 2.4: Algorithm GosCrash, Part II, code for processor $\mathbf{p}$.

□

Lemma 2.8 *If $f \leq n^{1-\varepsilon/3}$ then after Part I all non-crashed processors are informed.*

Proof: By the properties of $(|L|, 6, s)$-adaptive-communicator G_{AC}, in a set of correct processors there exists a subset $C \subseteq L$ of size at least $|L| - 2f$, that induces a subgraph of diameter at most $2k$.

This means that in each Mixing, every processors in C collects data from the whole set C. In particular, it gathers at least $|L| - 2f$ entries *active* in array *Active*. We show by induction that also in each phase each processor in C executes DistributedRequest:

In *Phase(0)* it is trivial, since $\frac{8n}{s^0} \geq n$.

Let us consider *Phase*$(i + 1)$ for $i \geq 0$. By assumption, in previous phase all processors in C sent messages to their neighbors in (n, s, s^i)-distributor G_{Di}. By the properties of G_{Di}, there are at most $\frac{8f}{s^i} \leq \frac{8n}{s^{i+1}}$ processors that are not neighbors of C in this graph. This is an upper bound on the number of rumors unknown in C at the beginning of *Phase*$(i + 1)$. After first Mixing in this Phase every processor in C lacks at most $\frac{8n}{s^{i+1}}$ rumors, which completes the induction.

In particular, in *Phase*(k) leaders use complete graph ((n, s, n)-distributor) in DistributedRequest. This means that messages are sent by set C to every processor at least once, and thus after first Mixing of the first informing phase all processors in C are informed. Consequently, after last DistributedRequest of the last informing phase, all correct processors are informed.

□

Lemma 2.9 *If there are at least $\frac{6n}{s^{i-1}}$ non-faulty processors at the end of Epoch(i) of Part II then all of them are informed.*

Proof: By the properties of (n, s, s^i)-communicator G_{Ci}, in this case there exists a center C of more than $\frac{3n}{s^{i-1}}$ processors and diameter at most $2k$, composed of processors that survive this epoch. Each processor in C gathers more than $\frac{3n}{s^{i-1}}$ values *active* at each step. We prove inductively that every processor in C runs DistributedRequest in each phase.

By construction, this is trivial for $j = i$;

Now, for any $j > i$, the set C has at least $n - \frac{3n}{s^{j-1}}$ neighbors in G_{Cj}, by expanding property (Theorem 2.3). If in *Phase*(i, j) requests were sent by whole set C then after this phase at most $\frac{3n}{s^{j-1}}$ rumors are not known in C. In first Mixing of *Phase*$(i, j + 1)$ every processor in C gathers all rumors known in C, which proves the claim.

In gathering *Phase*(i, k) each processor in C sends requests to all unknown processors - thus becoming informed. In informing *Phase*(i, k) each processor in C sends message to all uninformed processors - thus completing the gossip.

□

Lemma 2.10 *Each epoch in Part II contributes $O(n^{1+\varepsilon/3})$ to the message complexity.*

Proof: First we analyze Mixing subroutines. In *Epoch(1)*, mixing uses communicator of degree s, so there are $O(ns)$ messages sent.

Consider *Epoch*(i), for $i > 1$. It follows from Lemma 2.9 that at the beginning of this epoch either all processors are already informed, or at most $\frac{6n}{s^{i-2}}$ processors are non-faulty. In the first case no messages are sent. In the second case we use communicator of degree s^i, so there are at most $s^i \cdot \frac{6n}{s^{i-2}} = 6ns^2$ messages sent in each round.

To count the number of requests, we first observe that in *Phase*(i, i) the number of request is no greater than the number of mixing messages, since the same communicator is used. Now consider *Phase*(i, j) for $j > i$. We now claim that all requests are sent to a set of at most $\frac{3n}{s^{j-2}}$ processors.

Let $\mathbf{p}$ be any processor that sends requests, and let $U_{\mathbf{p}}$ be a set of processors that were *unknown* to $\mathbf{p}$ after the first Mixing in this phase. Since $\mathbf{p}$ marked itself *active*, $|U_{\mathbf{p}}| \leq \frac{3n}{s^{j-2}}$. We now prove that all requests in this phase are sent only to processors in $U_{\mathbf{p}}$.

For any processor x, let A_x denote a set of processors with value *active* in array $Active_x$ after the third Mixing in this phase. Now, assume that some $\mathbf{p}'$ sends requests. By properties of communicator, both sets $A_{\mathbf{p}}$ and $A_{\mathbf{p}'}$ contain more than half of correct nodes. Thus there exists $\mathbf{q} \in A_{\mathbf{p}} \cap A_{\mathbf{p}'}$. Since $\mathbf{q}$ was before the third Mixing in distance at most $2k$ from $\mathbf{p}$, it received all rumors outside $U_{\mathbf{p}}$ during the second Mixing. In the third Mixing it transmitted this rumors to $\mathbf{p}'$, which proves the claim.

Combining the claim with the fact that each processor may receive at most s^j requests, we get that the total number of requests in *Phase*(i, j) is $\frac{3n}{s^{j-2}} \cdot s^j = O(ns^2)$. Since the number of phases is constant, this implies that the total number of requests in considered *Epoch*(i) is also $O(ns^2)$.

The total number of answers is at most as large as the total number of requests. The same analysis applies to informing phases, showing that the number of messages sent in these phases is not bigger than in gathering phases. Thus total communication complexity is $O(ns^2)$.

□

Lemma 2.11 *Message complexity of Part II is $O(fn^{\varepsilon})$.*

Proof: Note that if $f \leq n^{1-\varepsilon/3}$, all non-crashed processors are informed before Part II by Lemma 2.8, thus no message is sent during Part II. In the remaining case $f > n^{1-\varepsilon/3}$, we combine the result obtained in Lemma 2.10 with the fact that the number of epoches is constant. We upper-bound the total number of messages by $O(1) \cdot O(n^{1+\varepsilon/3}) = O(fn^{\varepsilon})$.

□

Combining Lemmas 2.7 and 2.11 we obtain the final result.

Theorem 2.12 *For any n there exists an $(n-1)$-crash resilient gossip algorithm working in constant time and using $O(n+fn^{\varepsilon})$ messages when f failures occur.*

2.3.4 Time vs. messages trade-off

Algorithm presented above works correctly for any given ε. In particular, there is no requirement for ε to be constant. We can use for example $\varepsilon = \frac{1}{\log n}$. In this case, the asymptotic cost must be computed more carefully, as some constant factors that were ignored so far, become non-constant. A more precise analysis of above algorithm leads to the following theorem.

Theorem 2.13 *For any $s \geq 2$, there exists a $(n-1)$-crash-resilient gossip algorithm that works in time $O(\log_s^3 n)$ and uses $O(ns^2 \log_s^3 n)$ messages.*

Proof: Consider the algorithm from Figure 2.4, executed for a given parameter s and $k = \lceil \log_s n \rceil$. There are $O(\log_s^3 n)$ rounds of computation, and in each round there are at most ns^2 messages used, which yields the aforementioned total complexity.

□

2.4 Omission failures

This section is devoted to gossiping with omission failures. We start by reminding the reader that since omission-faulty processor may in particular omit all messages starting from some given round, the model with omission failures is at least as hard to handle as the one with crashes. In this section we prove that from the point of view of communication complexity it is substantially harder. Namely, communication complexity of any t-omission-resilient algorithm must be $\Omega(n + tf)$.

Observe that in case of omission failures, when some processor **p** expects a message from some other processor **q** and does not receive it, there is more than one possibility. Either **q** or **p** or both processors are faulty. If processor **p** informs other processors that **q** is faulty, this may be false information, if **p** is faulty itself. Note that such a misleading message is not possible in the case of crash failures, because if **p** was faulty, it could not send any message. In the case of omission failures, processor **p** may inform others only that **q** failed to communicate with it. A correct processor may fail to communicate with up to t processors. Only when some processor fails to communicate with at least $t+1$ processors, we know for sure that it is faulty. We use this to prove lower bound on message complexity of gossip.

2.4.1 Lower bound

Theorem 2.14 *Every t-omission-resilient gossip algorithm must use at least $\Omega(n + ft)$ messages when $f \leq t$ omission failures occur during the execution.*

Proof: Consider a t-resilient gossip algorithm $\mathbf{P}$. The lower bound $\Omega(n)$ is obvious from the fact that in an execution without failures each processor must send and receive at least one message.

Consider now execution with f omission failures. We show that there exist some failure patterns that force $\mathbf{P}$ to send more than $\frac{ft}{4}$ messages.

The idea is as follows: We introduce some set of processors that do not receive any message, either because they are faulty, or because senders of these messages are faulty. We show that in order to determine if any of these processors is correct, system must sent $\Omega(t)$ messages to every single processor in this set. If this set is of size $\Theta(f)$, it requires $\Omega(ft)$ messages. The rest of the proof is a formalization of this intuition.

Consider two following classes of failure patterns:
We call a failure pattern X-*passive*, if:

> X is a set of faulty processors and in every round all processors in X omit all received messages.

We say that a failure pattern is $(A, B, \mathbf{q})$-*semi-passive*, where A and B are disjoint sets and $\mathbf{q} \notin A \cup B$, if:

> $A \cup B$ is a set of faulty processors and in every round all processors in A omit all received messages, and all processors in $A \cup B$ omit all messages sent to processor $\mathbf{q}$.

Claim: If algorithm $\mathbf{P}$ sends less than $\frac{ft}{2}$ messages during an execution with any X-passive failure patterns with $f = |X| \leq \frac{t}{2}$ failures then there exists an $(A, B, \mathbf{q})$-semi-passive failure pattern with at most t failures such that the execution of $\mathbf{P}$ with this failure pattern is not valid (gossiping is not solved).
Proof of Claim: Assume that there exists an X-passive failure pattern with $f = |X| \leq \frac{t}{2}$, and execution $\mathcal{E}_1$ in which $\mathbf{P}$ sends less than $\frac{ft}{2}$ messages. It means that there exists some processor $\mathbf{q} \in X$ to which there were less than $\frac{t}{2}$ messages sent. Consider now $(A, B, \mathbf{q})$-semi-passive failure pattern, where $A = X \setminus \{\mathbf{q}\}$ and the set B contains all processors not from X that sent a message to $\mathbf{q}$ during execution $\mathcal{E}_1$. Note that $|B| \leq \frac{t}{2}$, by the choice of $\mathbf{q}$. Let $\mathcal{E}_2$ be the execution of algorithm $\mathbf{P}$ with the considered $(A, B, \mathbf{q})$-semi-passive failure pattern. By induction on the number of rounds, execution $\mathcal{E}_2$ is well-defined and executions $\mathcal{E}_1, \mathcal{E}_2$ of algorithm $\mathbf{P}$ are indistinguishable from the point of view of any processor: in both executions no processor in $A \cup \{\mathbf{q}\}$ receives any message, while the remaining non-faulty processors receive all

messages addressed to them. Processor $\mathbf{q}$ does not receive any message in both executions although it should receive at least one in execution $\mathcal{E}_2$ where it is not faulty. Hence the execution $\mathcal{E}_2$ is incorrect. Note that number of faulty processors in this execution is $|A \cup B| \leq |X| - 1 + \frac{t}{2} < t$ which violates t-omission-resiliency of the algorithm.

It follows from the Claim that in all executions against any X-passive failure patterns (with $f \leq \frac{t}{2}$ faulty processors), protocol $\mathbf{P}$ must use at least $\frac{ft}{2}$ messages. Consequently, for $f \leq t$, $\mathbf{P}$ must use at least $\frac{ft}{4}$ messages.

□

It is worth mentioning that unlike in crash failures, Theorem 2.14 holds for *all* t-omission-resilient gossiping algorithms, not only for those terminating in constant time.

2.4.2 Algorithm

The main idea behind the algorithm is as follows: every correct processor may fail to communicate with up to t processors. In order to identify any processor as a faulty one, we need only to get confirmation from at least $t+1$ processors that failed to communicate with it. We will perform this using list *BrokenLinks* to store pairs of processors that failed to communicate. Initially this list is empty. Whenever processor $\mathbf{p}$ sends a request to $\mathbf{q}$ and does not receive reply, it adds pair $(\mathbf{p}, \mathbf{q})$ to this list. This list is send in each message. Processor $\mathbf{p}$ at the beginning of each round scans received messages for new pairs and adds them to list *BrokenLinks*. Then it marks $Rumor_{\mathbf{p}}[\mathbf{q}] = \mathit{faulty}$ for each $\mathbf{q}$ that occurs in more than t pairs in $BrokenLinks_{\mathbf{p}}$.

Let L be a set of first $2n^{\frac{5}{6}}$ nodes, T be a set of first $\min\{6t+1, n\}$ nodes, A be the set of all nodes. Let G_A be a complete graph. Let G_D be a predefined $(n, n^{\frac{1}{6}}, 8)$-distributor with L being the set of leaders. Let G_{C0} be a predefined $(|L|, n^{\frac{1}{6}}, n^{\frac{1}{6}})$-adaptive-communicator on set L, and let G_{C1} be a predefined $(n, 6, n^{\frac{1}{3}})$-adaptive-communicator on the set of all processors.

Pseudocode of the algorithm is presented on Figure 2.5.

2.4.3 Analysis

Lemma 2.15 *Algorithm* GosOmission *terminates correctly in constant time and has message complexity* $O(n + ft)$.

Proof: Constant time complexity of algorithm GosOmission follows directly from the pseudo-code, since each subroutine has constant number of rounds.

We prove correctness of the algorithm together with its message complexity, because for both proofs we need to consider the same three cases. We start by

1. *Small mixing 1*:
   - If $\mathbf{p} \in L$, run **DistributedRequest**(G_D, all)
     else, wait for 3 rounds.
   - Run **Mixing**$(G_{C0}, 12)$.
   - If $\mathbf{p} \in L$, run **DistributedRequest**(G_D, all)
     else, wait for 3 rounds.
2. If $\mathbf{p}$ knows at most $n - 2n^{\frac{2}{3}}$ rumors, run **Mixing**$(G_{C1}, 6)$
   else, wait for 6 rounds.
3. If $\mathbf{p} \in T$, and $\mathbf{p}$ knows at least $n - 6t$ rumors, run **DistributedRequest**$(G_A, unknown)$
   else, wait for 3 rounds.
4. *Small mixing 2*:
   - If $\mathbf{p} \in L$, run **DistributedRequest**(G_D, all)
     else, wait for 3 rounds.
   - Run **Mixing**$(G_{C0}, 12)$.
   - If $\mathbf{p} \in L$, run **DistributedRequest**(G_D, all)
     else, wait for 3 rounds.
5. If $\mathbf{p}$ is informed, run **Mixing**$(G_{C1}, 6)$
   else, wait for 6 rounds.
6. If $\mathbf{p}$ is uninformed, send messages to all processors in T.
7. Wait for 2 rounds and finish.

Figure 2.5: Algorithm GosOmission, code for processor $\mathbf{p}$.

reminding that messages sent as the replies do not influence the communication complexity. It is enough to estimate the number of other messages.

Each Mixing$(G_{C0}, 12)$ uses $O(n)$ messages, because graph G_{C0} has $2n^{\frac{5}{6}}$ nodes and degree $n^{\frac{1}{6}}$. Also each DistributedRequest(G_D, all) uses $O(n)$ messages, because graph G_D has $O(n)$ edges. For the remaining subroutines we consider three cases:

Case $f \geq \frac{n}{6}$: Here $O(n + ft) = O(n^2)$, so bound is trivially met. Correctness comes from the fact that all processors are in the set T, and therefore

all run DistributedRequest in point 3. After receiving the replies, all become informed.

Case $tf \geq n^{\frac{4}{3}}$: Mixing$(G_{C1}, 6)$ uses at most $n^{\frac{4}{3}}$ messages, because degree of graph G_{C1} is at most $n^{\frac{1}{3}}$. In this case, however, $n^{\frac{4}{3}} = O(tf)$, so bound is met. Only DistributedRequest$(G_A, unknown)$ and requests in point 6 may violate complexity bound, potentially using $\Omega(nt)$ messages. To prove that this is not the case, observe first that in graph G_{C1} there exists a subset C of $n - 2f$ correct nodes that induces subgraph of diameter at most 6. It means that during the first Mixing$(G_{C1}, 6)$, each processor in C gathers rumors of the whole set C, thus lacks at most $2f$ rumors. Therefore in DistributedRequest$(G_A, unknown)$:

- processors from $T \cap C$ (at most $6t + 1$) send at most $2f$ messages each,
- processors from $T - C$ (at most $2f$) send at most $6t$ messages each.

Number of messages in this line is therefore $O(ft)$. Moreover, $T \cap C$ has at least one processor, and it becomes informed after point 3.

Now, since $|T \cap C| \geq 6t+1-2f \geq t+1$, during the second Mixing$(G_{C1}, 6)$, each processor in C gathers for each processor either its rumor or at least $t + 1$ confirmations of its failure. Thus every processor in C is informed after this subroutine. It means that before point 6 at most $2f$ processors are not informed. Those processors send messages to set T in point 6, and become informed after receiving replies. Total number of messages is $O(ft)$.

Case $f \leq n^{\frac{2}{3}}$: Here we focus on set L. Let f_1 be a number of failures in graph L. In graph G_{C0}, there exists a set C of at least $|L| - 2f_1$ correct nodes that induces a subgraph of diameter at most 12. By properties of $(n, n^{\frac{1}{6}}, 8)$-distributor, this set has at least $n - f_1$ neighbors in graph G_D. At least $n - f_1 - f \leq n - 2f$ of them are non-faulty. Let us denote this set by X. In each *Small mixing*, each two processors in X exchange their knowledge: first sending its rumor to some processor in C then routing this rumor through the set C in Mixing subroutine, and finally by transmitting it from some processor in C. It means that:

- After *Small mixing 1*, at most $2f$ processors run Mixing$(G_{C1}, 6)$. Number of messages used is $O(n^{\frac{1}{3}}) \cdot 2f = O(n)$.
- In DistributedRequest$(G_A, unknown)$:
  - processors from $T \cap X$ (at least 1, at most $6t + 1$) send at most $2f$ messages each,
  - processors from $T - X$ (at most $2f$) send at most $6t$ messages each.

therefore at most $O(ft)$ messages are used.

- After *Small mixing 2*, at most $2f$ processors are uninformed, because each processor in X receives either a rumor or $t+1$ confirmations of faults for every processor.

- At most $2f$ processors run second Mixing$(G_{C1}, 6)$, using again $O(n^{\frac{1}{3}}) \cdot 2f = O(n)$ messages.

- In point 6 all uninformed processor send messages to T. Since there is at least one correct informed processor in this set, they receive replies and become informed. There are $O(ft)$ messages, as in the previous case.

Summarizing, the message complexity of the algorithm is $O(n + ft)$, which yields a contribution $O(t)$ to the message complexity by each omission failure.

□

Therefore we proved the final result.

Theorem 2.16 *For any $t < n$, there exists a t-omission-resilient gossip algorithm working in constant time and using $O(n + tf)$ messages when f failures occur.*

2.4.4 Authenticated Byzantine failures

The approach presented for omission failures works efficiently for the gossip problem in message-passing system with authenticated Byzantine failures.

The intuition behind adapting our result for omission failures to authenticated Byzantine failures model is that the information is propagated correctly in the latter model, due to the authentication property, provided it is not omitted in transfer and the source of the rumor is not faulty This suffices for adapting the algorithm designed against omission failures to the setting with authenticated Byzantine failures, with the same asymptotic time and message complexity (see [63] for detailed discussion of a simulation of omission faults on the top of restricted Byzantine faults, including authentications).

2.5 Byzantine failures

2.5.1 Lower bound

Theorem 2.17 *Every t-Byzantine-resilient gossip algorithm, where $0 < t < n$, has message complexity $\Omega(nt)$, even in the executions without failures.*

Proof: We first prove the theorem for $t < \frac{n}{3} - 1$. Then we extend this result for all $t < n$.

Assume, to the contrary, that there exists a protocol $\mathbf{P}$ that in any execution without failures sends less than $\frac{nt}{6}$ messages. It means that in any execution there is a set of more than $\frac{2n}{3}$ processors that communicate with at most $\frac{t}{2}$ processors each.

Let A be a set of first $t + 1$ processors and B be the set of remaining processors. We now consider two initial settings: A^0, in which all processors in A have rumors equal 0, and A^1, in which all processors in A have rumors equal 1. In both settings all processors in B have rumors equal 0. By our assumption, there exists more than $\frac{n}{3}$ processors that in both settings communicates with at most $\frac{t}{2}$ other processors. At least one of them is outside A, because $t < \frac{n}{3}$. Let call this processor $\mathbf{p}$.

Denote by T a set of processors that $\mathbf{p}$ communicates with in configuration A^0 when there are no failures, and by T' a set of processors that $\mathbf{p}$ communicates with in configuration A^1 when there are no failures. Now consider an execution with initial configuration A^0, when all processors in $T \cup T'$ are faulty and behave as follows:

- towards processor $\mathbf{p}$ - as in execution with configuration A^1 and without any failures
- towards remaining processors - as in execution with configuration A^0 and without any failures

Observe that from the point of view of processor $\mathbf{p}$ this execution is equivalent to execution in configuration A^1 without any failures, and from the point of view of any other correct processor this execution is equivalent to execution in configuration A^0 without any failures. Consequently, processor $\mathbf{p}$ will end without correct rumors of all processors in set A. Since $|A| = t + 1$, at least one of these processors is correct, which violates gossip requirements. This completes proof for $t < \frac{n}{3} - 1$.

Consider now any $t \geq \frac{n}{3} - 1$. By definition, t-Byzantine-resilient algorithm for such t is also $(\frac{n}{3} - 2)$-Byzantine-resilient. We have already proven that such algorithm has message complexity $\Omega(n^2)$. Therefore for each $t < n$ bound $\Omega(nt)$ also holds.

□

2.5.2 Algorithm and analysis

Last piece of analysis of gossip in different failure settings is to present an algorithm that matches lower bound. This was already done by Dolev and Reischuk in [29]. Although this algorithm is quite natural and very simple, it

may also be viewed as one following the general framework (slightly simplified) described in Section 2.2. Here we present it in such way.

Theorem 2.18 [29] *For any $t < n$, there exists a constant-time t-Byzantine-resilient gossip algorithm working in constant time and using $O(nt)$ messages.*

Proof: The algorithm distinguishes between two cases.

If $2t \geq n$ then each processor broadcasts its rumor to all other processors; message complexity in this case is $O(n^2) = O(nt)$.

If $2t < n$ then let T be a set of first $2t + 1$ processors. In the first round every processor sends message to all processors in T. In the second round every processor in T receives messages, updates its local memory and sends messages to all processors. In the third round every processor receives messages and for every other processor $\mathbf{q}$ it sets the rumor of $\mathbf{q}$ to those which occurs in the majority of received messages. Since $f \leq t$, at least $t + 1$ processors in T are correct, therefore this procedure guarantees selecting a correct rumor of $\mathbf{q}$, unless $\mathbf{q}$ is faulty. The message complexity in this case is $O(nt)$.

□

2.6 Open problems

Above analysis determines the impact of faulty processors on the gossiping problem. In the class of constant time algorithms the question is answered, as well as for arbitrary time for omission and Byzantine faults. The only part not proven to be optimal is crash resiliency in time longer than constant.

It is shown that for $t < n^{1-\varepsilon}$, gossiping can be performed in constant time and using $O(n)$ messages. For bigger t, we can trade time for message complexity. But this technique does not give linear message complexity for any time. The problem of gossiping with $O(n)$ messages for $t = n - 1$ remains open, even if we consider exponential execution time.

Above framework is created to work in synchronous setting, but it might be possible to extend it to work in partially synchronous setting of some kind [33], with a little loss of efficiency. We note here that resiliency to faults in completely asynchronous distributed setting is somewhat tricky concept. In consensus problem it is known that even one crash makes it unsolvable. In gossiping, it is not clear even how to define the problem. When messages from one processor travels long enough, at some point rest of the system should finish, not knowing if that processor crashed or not.

One interesting problem which was not address in this thesis is a cost of sequence of gossip procedures in case of omission failures. Imagine that we need to perform gossip many times, say n, on the same set of nodes. This is classical case when nodes gather some data and have to exchange it periodically. In

case of crash failures, each faulty machine will immediately be identified by its neighbors and removed from the computation. In case of Byzantine failures, we have seen that we need to pay additional cost each time, even when all machines are correct. In case of omission failures the problem remains open: although faulty machines are increasing the cost of computation only when they are acting incorrectly, they may remain undetected. For example, when there are two sets of $\frac{t}{2}$ machines that looses each message sent between each other, they may add $\Omega(t^2)$ to cost of each gossip, without revealing which of these sets is faulty. Effective algorithm should therefore change the communication pattern to isolate nodes that are acting suspiciously. Observe that simple removing of each broken link from future executions (in a sense that machines does not try to communicate with machines that did not answer before) is not sufficient.

Above algorithms open also a way to analyze gossip in more sophisticated settings, i.e. in window-generated failures (when the number of failures in a given number of round is bounded) or general network topologies.

Chapter 3

Consensus

3.1 Introduction

Consensus is one of the fundamental problems in distributed computing. Basically, it encompasses all tasks when all units have to agree on one common decision.

Canonical introduction of the consensus problem is a *Byzantine Generals Problem* [58]. There is a story of a generals from Byzantium that besiege an enemy city. There is a dispute if the army should attack or not. They can not leave their cohorts, in case of retaliation attack from the city. They can only communicate by messengers. The problem is that some of the generals are traitors allied with the city. Their goal is to prevent the siege from success. It can be done in two ways. First possibility is that loyal generals become divided and some of them attack and some don't. Second possibility is that the army makes a decision that is against the intention of all loyal generals (attack when no one initially votes for that or withdraw when all voted for attack). In both cases traitors win.

It is known that if among n generals there are at least $\frac{n}{3}$ traitors, they will always be able to trick the rest and win. If the number of traitors is smaller, there are algorithms to ensure correct decision.

In the original definition, traitors may behave in any possible way to break the siege. But the main problem remains nontrivial also in more restricted setting. In particular, even when the only possible violation of protocol is the death of general (which corresponds to crash failure in our definition), still every deterministic solution requires at least $f + 1$ rounds of communication [2]. Even though it has been extensively studied in various settings for the last three decades, still not much is known about the communication complexity of this problem, for example how many bits need to be sent by processors until decisions are made.

We start this chapter with proper formalization of the consensus problem,

together with an outline of the previous and new solutions. Then in Section 3.2 we define a concept of *well connected majority*, and prove the existence of graphs that we will use in our algorithms. In Section 3.3, we present a deterministic algorithm and its analysis. Section 3.4 is devoted to the quantum consensus. We start with an introduction to quantum model of computation and formalization of quantum distributed system. Then we present an algorithm and its analysis. We end this chapter with discussion of some open problems.

3.1.1 Problem setting

In the literature, consensus is defined by three properties:
Termination: Each processor eventually chooses a decision value, unless it crashes,
Agreement: No two processors choose different decision values,
Validity: Only a value among the initial ones may be chosen as the decision value.

Definition of the problem in our formalization is the following: In the initial state, each machine M_i has some initial value v_i, written on its working tape. Following the Byzantine Generals Problem, we assume that this is one bit value (*attack* or *withdraw*). It is known that solutions for this case may be extended to multi-value consensus with an extra logarithmic factor in communication cost, using the tournament technique (see e.g., [43] for details).

The goal is to choose one of the initial values. Formally, we say that protocol P successfully solves consensus, if for any sequence $< v_1, v_2, \ldots, v_n >$ of initial values, in accepting state there exists a $v = v_i$ for some i and all unit have somewhere on its working tape the same sequence:

$Decision = v$,

and symbol *Decision* is written exactly once on this tape. In a faulty setting, a set F of faulty machines is introduced. We say that protocol P successfully solves consensus, when every correct machine ($M \in A \backslash F$) in accepting state has on its working tape the sequence:

$Decision = v$.

We say that protocol P is t-resilient, if for every set of faulty machines F such that $|F| \leq t$, P successfully solves consensus. Depending on type of failures, we obtain definitions of t-crash-resiliency, t-omission-resiliency and t-Byzantine-resiliency.

3.1.2 Previous and related work

The problem of consensus was introduced by Pease, Shostak and Lamport [65]. They showed [58, 65] that number t of faulty processors needs to be smaller

than $\frac{n}{3}$ for a solution to exist, assuming synchrony and Byzantine faults.

Fisher, Lynch and Paterson [37] showed that the problem cannot be solved deterministically in the asynchronous message-passing setting even if only one processor may crash. On the other hand, randomized solutions exists for the number of crashes up to $\frac{n}{2}$ (see e.g. [14]). One may find several other possible solutions in asynchronous model in [25].

Fisher and Lynch [36] showed that a t-crash-resilient deterministic solution to consensus requires $t+1$ rounds. The lower bound for communication complexity in this model is $\Omega(n)$ [5]. For the more severe kinds of process failures, like omissions or Byzantine, the lower bound $\Omega(n+t^2)$ on the communication complexity was proved by Dolev and Reischuk [29]. On the other side, Garay and Moses [39] developed an algorithm with polynomial-size messages and operating in $t+1$ rounds, for $n > 3t$ processors subject to Byzantine failures (although the obtained polynomial was large).

Dolev and Reischuk [29] studied the message complexity of consensus in the case of Byzantine faults. They distinguished between pure Byzantine faults and a less demanding situation when some (cryptographic) authentication mechanism is available, which makes forging of forwarded messages impossible. They showed 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 showed that any algorithm with authentication needs to send $\Omega(n+t^2)$ messages, and that achieving the goal with this number of messages is possible. This proves that, in the case of malicious faults, the required number of messages has to be quadratic in n.

Surprisingly, only a trivial linear lower bound $\Omega(n)$ [5] on the number of messages is known for crash failures. Dwork, Halpern and Waarts [32] designed an algorithm with $O(n \log n)$ communication bits, but their algorithm worked in exponential time. Galil, Mayer and Yung [38] improved this solution to $O(n)$ communication bits, but still in the cost of exponential time. They also developed the first time-efficient algorithm with subquadratic number of communication bits, equal $O(\frac{n^2}{\log n})$.

Randomized solutions to consensus are much more efficient that the classical ones. Bar-Joseph and Ben-Or showed that in randomized case it may be solved in $O(\frac{t}{\sqrt{n \log n}})$ expected number of rounds, and this is an optimal solution [13]. Ben-Or showed also that consensus in a quantum model can be solved in constant number of rounds [15]. Both this solutions use all-to-all communication in every round, therefore their message complexity is high.

Relevance of the consensus problem to fault-tolerant broadcast and other communication problems was discussed by Hadzilacos and Toueg [47]. A related problem of almost-everywhere agreement was considered by Dwork, Peleg, Pippenger and Upfal [34] and later by Upfal [73]. In this model an agree-

ment can be reached by a fraction of processors when a smaller fraction is crashed, even in some constant-degree underlying networks.

3.1.3 New results

We study this problem in a message-passing synchronous system with crash-prone processors. We develop deterministic consensus that works in time $O(t)$ and uses only $O(n \log^2 n)$ bits, for less than $\frac{n}{3}$ crashes. It can be contrasted with algorithms under omission and Byzantine failures, where quadratic number of bits is necessary. We apply this technique for sensing the size of local neighborhood in specific graphs, to decrease communication complexity of quantum consensus algorithm, from quadratic to $O(n \log^3 n)$.

3.2 Well-connected majority

Many consensus algorithms are designed on the idea that a majority of non-faulty processors decide, and then they spread its decision to the remaining processors. The real challenge in this approach is how to define a majority that has enough processors to decide. For example, if such majority is predefined in the algorithm, sufficient number of crashes in this set may lead to failure of the whole protocol. It might be possible to detect such event and repeat this procedure with different set, but this leads to big communication overhead of many repetitions.

A better result can be achieved by using special communication pattern that only guarantees that a majority exists, not specifying it directly. In this approach every processor must recognize whether it belongs to a decision-making majority or not, based on its local knowledge. The previous instantiations of this method [19, 20] suffered from the fact that they used gossiping as a subroutine, which still requires a quadratic number of bits to be sent (although the number of messages may be reduced to $O(n \text{ polylog } n)$, comparing to $\Omega(n^2)$ of the naive approach with pre-defined majority).

Our approach allows us to recognize whether a processor is in a compact majority component by exchanging only polylogarithmic number of bits per processor, on average. To achieve this, we use graphs with specific properties, in which nodes are identified with processors, and each processor sends messages only to the neighbors in these graphs. The precise schedule of how to use the graphs will be defined in Section 3.3, here we only define graphs and prove their existence.

We start with some additional definitions, extending the framework from Chapter 2. For given undirected graph $G = (V, E)$ and set $W \subseteq V$, we call the subgraph of G containing only nodes in W and all edges with both ends in W a *subgraph of G induced by set W* and denote it by $G|_W$.

For a given subset of edges $E' \subseteq E$, let $V(E')$ denote the set of nodes incident to some edge in E'. We call edge $\{v, w\}$ *internal* for a given set of nodes $B \subseteq V$ if $v, w \in B$.

By $N_G^i(A)$ we denote the set of nodes that are of distance at most i from some node in A in graph G. We say that a set of nodes $C \subseteq V$ is k-*dense* in graph G if each node $v \in C$ has at least k neighbors in C.

Let γ be a constant to be specified later. Consider an undirected graph $G = (V, E)$ of n nodes, and a positive integer δ. We are interested in the following graph properties.

δ-Dense-Compact-Subgraph (δ-DCS): For every set $B \subseteq V$ of at least $\frac{2n}{3}$ nodes there exists a δ-dense subset $C \subseteq B$ of size at least $\frac{n}{2}$ and diameter at most $\gamma \log n$.

δ-Edge-Density (δ-ED): For any set A of at most $\frac{n}{2}$ nodes, the total number of internal edges is at most $\frac{7|A|\delta}{4}$.

Theorem 3.1 *For every n, $\delta > 288 + 2\log n$ and $\gamma > 23$, there exists a graph of n nodes, each of degree between 2δ and 4δ, satisfying properties δ-DCS and δ-ED.*

Proof: Let $G = (V, E)$ be a random graph of n-nodes, with the probability $\frac{6\delta}{n}$ of each pair $\{v, w\}$ to be in E.

Expected number of edges is $\binom{n}{2}\frac{6\delta}{n}$. By Chernoff bound, probability that it has more than $2 \cdot \binom{n}{2}\frac{6\delta}{n}$ is at most 2^{-n}. Since $2 \cdot \binom{n}{2}\frac{6\delta}{n} \geq 6\delta n$, probability that this graph has at most $6\delta n$ edges is at least $1 - 2^{-n}$. Moreover, each node has degree at least 2δ and at most 4δ with probability at least $1 - 2^{1-\delta/9}$, by the similar argument. We first prove that it satisfies the following density-kind properties with high probability, and then we argue that these properties guarantee δ-DCS and δ-ED.

Property $\mathcal{A}$. For any set A of at most $\frac{n}{12}$ nodes, the number of nodes with more than δ neighbors in A is less than $\frac{n}{24}$.

Property $\mathcal{B}$. For any set B of at least $\frac{2n}{3}$ nodes, the number of nodes with less than 2δ neighbors in B is less than $\frac{n}{24}$.

Property $\mathcal{C}$ (equivalent to property δ-ED). For any set A of at most $\frac{n}{2}$ nodes, the total number of internal edges is at most $\frac{7|A|\delta}{4}$.

Proof of Property $\mathcal{A}$. Note first that it is sufficient to consider only sets A of size exactly $\frac{n}{12}$, since a node that has more than δ neighbors in set A has also more than δ neighbors in any superset $A' \supseteq A$ of size exactly $\frac{n}{12}$;

Let $A \subseteq V$ be a set of $\frac{n}{12}$ nodes, and $X_A \subseteq V$ be the corresponding set of nodes with more than δ neighbors in A. Every v has expected number $6\delta \cdot \frac{n}{12}/n = \frac{\delta}{2}$ neighbors in A. The probability that v has $N > \delta$ neighbors in

A is bounded, using Chernoff bound, by:

$$\mathbf{Pr}\left(N > (1+1)\cdot\frac{\delta}{2}\right) < \left(\frac{e}{2^2}\right)^{\frac{\delta}{2}} < 2^{-\frac{\delta}{4}} .$$

Thus, the probability that there exists a set A of size $\frac{n}{12}$ and the corresponding set X_A with at least $\frac{n}{24}$ nodes is at most:

$$\binom{n}{n/12}\binom{n}{n/24}(2^{-\delta/4})^{n/24} < 2^n \cdot 2^n \cdot 2^{-3n} = 2^{-n}$$

since $\delta > 288$.

Proof of Property $\mathcal{B}$. Note first that it is sufficient to consider sets B of size exactly $\frac{2n}{3}$, because each node with less than 2δ neighbors in set B has also less than 2δ neighbors in an arbitrary subset $B' \subseteq B$ of size exactly $\frac{2n}{3}$.

Let B be a subset of $\frac{2n}{3}$ nodes. Let X_0 be the set of nodes with less than 2δ neighbors in B. Every v has expected number $6\delta \cdot \frac{2n}{3}/n = 4\delta$ neighbors in B. Probability that v has $N < 2\delta$ neighbors in B is bounded as follows, using again Chernoff bounds:

$$\mathbf{Pr}\left(N < (1-1/2)4\delta\right) < e^{-4\delta/8} = e^{-\delta/2} .$$

The probability that there exists a set B such that the corresponding set X_0 has $\frac{n}{24}$ elements is at most:

$$\binom{n}{2n/3}\binom{n}{n/24}(e^{-\delta/2})^{n/24} < 2^n \cdot 2^n \cdot 2^{-3n} = 2^{-n} ,$$

since $\delta > 144$.

Proof of Property $\mathcal{C}$. The proof is similar to the one for Property $\mathcal{A}$. Let $A \subseteq V$ be a set of a nodes, where $a \leq \frac{n}{2}$. We may assume $a \geq 7\delta/4$, since otherwise the total number of internal edges is at most $|A|^2 < 7|A|\delta/4$ with probability 1.

The probability that there is an edge between any two nodes in set A is $\frac{6\delta}{n}$, thus the expected number of internal edges is $\frac{a(a-1)}{2} \cdot \frac{6\delta}{n} = \frac{3a(a-1)\delta}{n}$. Consequently, the probability that the number N of internal edges is bigger than $7a\delta/4$ can be bounded using Chernoff bound:

$$\mathbf{Pr}\left(N > (1+\alpha)\cdot\frac{3a(a-1)\delta}{n}\right) < \left(\frac{e^\alpha}{(1+\alpha)^{1+\alpha}}\right)^{\frac{3a(a-1)\delta}{n}}$$

where $\alpha = \frac{7n-12(a-1)}{12(a-1)}$.

We consider two cases:

1. For $a \leq n/6$, $\alpha > 2$, and therefore

$$\mathbf{Pr}\left(N > (1+\alpha)\cdot\frac{3a(a-1)\delta}{n}\right) < e^{-\frac{6n}{12a}\cdot\frac{3a(a-1)\delta}{n}} < 2^{-a\delta}$$

2. For $a > n/6$ and $a < n/2$, $\alpha > 1/6$ and we use bound

$$\mathbf{Pr}\left(N > (1+\alpha)\cdot\frac{3a(a-1)\delta}{n}\right) < e^{-\frac{a^2\delta}{2n}}$$

Probability that there exists a set A of at most $\frac{n}{6}$ nodes and more than $\frac{7|A|\delta}{4}$ internal edges is at most

$$\sum_{a=7\delta/4}^{n/6}\binom{n}{a}\cdot 2^{-a\delta} \leq \sum_{a=7\delta/4}^{n/6}\left(\frac{ne}{a}\right)^a\cdot 2^{-a\delta} \leq$$

$$\leq \sum_{a=7\delta/4}^{n/6} 2^{a\log n}\cdot 2^{-a\delta} \leq \sum_{a=7\delta/4}^{n/6} 2^{-a\delta/2} \leq 2^{-7\delta/4} < \frac{1}{n}\ ,$$

since $\delta > 2\log n$.

Probability that there exists a set A of size between $\frac{n}{6}$ and $\frac{n}{2}$ and more than $\frac{7|A|\delta}{4}$ internal edges is at most

$$\sum_{a=n/6}^{n/2}\binom{n}{a}\cdot 2^{-\frac{a^2\delta}{2n}} \leq 2^n\cdot 2^{-2n} = 2^{-n}\ ,$$

since $\delta > 144$.

By the probabilistic argument, there exists a graph satisfying the three properties and such that each node has degree between 2δ and 4δ. The number of edges in this graph is obviously smaller than $3n\delta$. Property $\mathcal{C}$ is the same as property δ-ED. It remains to prove that this graph satisfies property δ-DCS.

Let us define, for any set of nodes X, the following operation:

$$F(X) = X \cup \{v \mid v \text{ has at least } \delta \text{ neighbors in } X\}$$

This operation is monotonic, and defined on a finite set, therefore it must have a fixed point. Also, starting from any set and iterating the operation F we will reach some fixed point in at most n steps. For any given B of size at least $\frac{2n}{3}$, we define a set $F^*(B)$ to be a fixed point obtained by iteration of F on B. We now prove that $|F^*(B)| < \frac{n}{12}$.

Let $X_0 = B$, and $X_{i+1} = F(X_i)$ for any i. By the Property $\mathcal{B}$, $|X_0| \leq \frac{n}{24}$. Let X_j be the first set in this sequence that has at least $\frac{n}{12}$ elements. From monotonicity of F we know that

$$X_i = X_{i-1} \cup \{v \mid v \text{ has at least } \delta \text{ neighbors in } X_{i-1}\}$$

Since both sets in the union have less than $\frac{n}{24}$ elements — the first by definition of X_i, and the second by the Property $\mathcal{A}$ — we end up with a contradiction.

Therefore, each set in this sequence has less than $\frac{n}{12}$ elements, and consequently $|F^*(B)| < \frac{n}{12}$.

To prove δ-DCS, for a given set $B \subseteq V$ of size at least $\frac{2n}{3}$ we define $C = B \setminus F^*(B)$. Each element in C has at least 2δ neighbors in B and at most δ neighbors in X, therefore it has at least δ neighbors in C. Moreover, $|C| \geq \frac{2n}{3} - \frac{n}{24} > \frac{n}{2}$. It remains to prove that C has diameter at most $2\log n$, which is guaranteed if we show an expansion property for graph $G|_C$: each set $A \subseteq C$ of size at most $\frac{|C|}{2} \leq \frac{n}{2}$ has at least $|A|/4$ neighbors in set $C \setminus A$. It follows from Property $\mathcal{C}$. More precisely, set A has at most $7|A|\delta/4$ internal edges, therefore there are at least $2|A|\delta - 7|A|\delta/4 = |A|\delta/4$ of edges between sets A and $C \setminus A$. Since degree of each node in graph G, and thus in $G|_C$, is at most 4δ, the number of nodes in $C \setminus A$ connected to set A is at least $\frac{|A|\delta/4}{4\delta} \geq |A|/16$. This yields diameter at most $2 \cdot \log_{17/16} n \leq 23 \log n$, and concludes the proof.

□

We have proven the existence of such graphs, but in order to use them in consensus algorithm, we need to prove their one more useful property. Let us take any graph $G = (V, E)$ with properties δ-DCS and δ-ED, and a node $v \in V$. Consider any set $S(v) \subseteq N_G^{\gamma \log n}(v)$, with the property that each node $w \in S(v) \cap N_G^{\gamma \log n - 1}(v)$ has at least 2δ neighbors in $S(v)$. We call set $S(v)$ *2δ-pseudo-dense* in graph G. (Prefix "pseudo" corresponds to the fact that there may be nodes in $S(v)$ of distance exactly $\gamma \log n$ from v that have less than 2δ neighbors in $S(v)$.) Observe that in particular if every node has degree at least 2δ, the set $N_G^{\gamma \log n}(v)$ is 2δ-pseudo-dense. Here we prove a lemma that tells something about size of such sets.

Lemma 3.2 *For any graph $G = (V, E)$ of n nodes, each of degree between 2δ and 4δ, satisfying property δ-ED, and for any node $v \in V$, any 2δ-pseudo-dense set $S(v)$ has at least $n/2$ nodes.*

Proof: We prove by induction on i, for $1 \leq i \leq \gamma \log n$, that $S(v) \cap N_G^i(v)$ has at least $\min\{(17/16)^i, \frac{n}{2}\}$ elements. For $i = 1$ it follows directly from the fact that $v \in S(v)$ has at least 2δ neighbors in set $S(v) \cap N_G^1(v)$. Suppose that the invariant $|S(v) \cap N_G^i(v)| \geq 2^i$ holds for $1 \leq i < \gamma \log n$, we prove it for $i+1$. If $S(v) \cap N_G^i(v)$ has size bigger than $\frac{n}{2}$, we are done. Otherwise set $A = S(v) \cap N_G^i(v)$ has at most $7|A|\delta/4$ internal edges. Consequently, there are at least $2|A|\delta - 7|A|\delta/4 = |A|\delta/4$ edges between sets A and $N_G^{i+1}(v) \setminus A$. Since each node in the second set has degree at most 4δ, the number of nodes in $N_G^{i+1}(v) \setminus A$ connected to set A is at least $\frac{|A|\delta/4}{4\delta} \geq |A|/16$. Therefore the size of set $S(v) \cap N_G^{i+1}(v)$ is at least $|A| + |A|/16 \geq |S(v) \cap N_G^i(v)| \cdot (17/16) \geq (17/16)^{i+1}$. We conclude that $S(v) \cap N_G^{\gamma \log n}(v)$ has at least $\min\{(17/16)^{\gamma \log n}, \frac{n}{2}\} \geq \frac{n}{2}$ elements.

□

In the algorithm we will also use a-expanding graphs, for $a = \frac{n}{2}, \frac{n}{4}, \ldots, 1$. *$a$-expanding graph* is a graph where each two subsets of size a are connected by an edge. It can be shown that there exist a-expanding graphs with maximum degree $O(\frac{n}{a} \log n)$, although the best explicit constructions reach only degree $O(\frac{n}{a}$ polylog $n)$ [74]. We will also use a $(n - 3t - 1, 3t + 1, d)$-disperser graph H_t, which is a simplified version of general disperser. This graphs is defined as a bipartite graph with the first $3t + 1$ processors on the right-hand side and the remaining $n - 3t - 1$ on the left-hand side, such that the left degree is d and each subset of the left-hand side of size $\frac{n-3t-1}{2}$ has more than $3t/2$ neighbors on the right-hand side. There exists a $(n-3t-1, 3t+1, d)$-disperser for $d = O(\log n)$, although, similarly as for expanding graphs, the best explicit constructions guarantee only $d = O($ polylog $n)$ [74].

3.3 Deterministic algorithm

In this section we present a deterministic consensus algorithm that tolerates $t < \frac{n}{3}$ crash failures. We start with defining data structures and variables used by the algorithm. We also describe the format of messages and the update rules based on received messages.

3.3.1 Local memory, messages and updates

Let $\delta = \max\{288, 2 \log n\} + 1$. Let $\{G_0, G_1, \ldots, G_{\log n}\}$ be a family of n-node graphs defined such that:

- G_0 satisfies δ-DCS and δ-ED properties (see page 53) and each node has degree between 2δ and 4δ.
- G_i, for $1 \leq i \leq \log n$, is a $\frac{n}{2^i}$-expanding graph of maximum degree $O(\frac{n}{2^i} \log n)$.

These graphs are known to all processors, with nodes identified with the processors.

Local memory. Processor $\mathbf{p}$ stores the following data:

- Value δ and all graphs $G_0, G_1, \ldots, G_{\log n}$.
- $v_{\mathbf{p}}$ (Value): a candidate value, 0 or 1; initialized into $initial\_value_{\mathbf{p}}$.
- $d_{\mathbf{p}}$ (Decision): a variable, of value *yes* or *no*, indicating whether a decision on the candidate value $v_{\mathbf{p}}$ has been already made;

Values *yes* and *no* are encoded as a single bits 1 and 0 respectively, but for the purpose of the presentation we will use the more informative text form (*yes* and *no*). We say that process $\mathbf{p}$ is *convinced* in a given round if it has $d_{\mathbf{p}} = yes$ at the end of this round.

Messages and memory update. Each message sent by processor $\mathbf{p}$ consists of two bits v, d, where bit v is equal to the current value of variable $v_{\mathbf{p}}$, and bit d is equal to the current value of variable $d_{\mathbf{p}}$. The algorithm has three phases. In each round, after receiving a set of messages, processor $\mathbf{p}$ performs the following update operations:

- In Phase 1 and 2: If some message received in this round contains $v = 1$, it sets $v_{\mathbf{p}} \leftarrow 1$.
- In Phase 3: If any message received in this round contains $d = yes$, it sets $d_{\mathbf{p}} \leftarrow yes$ and sets its value $v_{\mathbf{p}}$ to the value v received together with $d = yes$ (we say that $\mathbf{p}$ adopts a decision value and becomes convinced).

3.3.2 Algorithm

We first describe and analyze our algorithm for parameters n, t satisfying conditions $\frac{n}{12} \leq t < \frac{n}{3}$, later we show how to naturally extend it for any $t < \frac{n}{3}$. We call the algorithm *Majority-Sensing Consensus Algorithm*, or MSC for short, since the main idea behind it is as follows. First, in order to reduce the number of messages, each processor communicate only with its neighbors in some sparse graph. This graph guarantees that for every feasible execution there is a majority of processors that are "well-connected" by links of this graph. Second, most of processors in this majority recognize that they belong to it by exchanging a small number of short messages ("sensing neighborhood"). Then the majority makes a decision. Finally, remaining processors try to communicate with this majority and adopt the decision. They use family of expanding graphs, in order to make this fast and using small number of messages.

The algorithm consists of three phases.

Phase 1. Propagating values
The goal of this phase is to propagate decision values inside each connected component of graph G_0. This phase lasts $12t + 23 \log n$ rounds. Each processor $\mathbf{p}$ sends a message to all its neighbors in graph G_0 in the round just after setting $v_{\mathbf{p}}$ into 1. Note that a processor with initial value 1 sends it in the first round, while a processor that ends this phase with value 0 has not sent any message in this phase.

Phase 2. Sensing for a compact majority
In this phase processors probe local neighborhood with a message, to determine if they are connected to the majority set and then they make a decision. This phase lasts $23 \log n$ rounds. Each processor keeps sending a messages to all of its neighbors in graph G_0 until in some round it receives less than 2δ messages. If in the last round of this phase a processor $\mathbf{p}$ receives at least 2δ messages, it sets $d_{\mathbf{p}}$ to *yes* and sends a message to all its neighbors in graph G_0.

Phase 3. Calling for decision value.
The goal of this phase is for every processor to get to know the decision value. This phase lasts $2 \log n$ rounds. In this phase each processor that is not convinced yet, sends requests to other processors to learn about a decision value. More precisely, a processor $\mathbf{p}$ that has $d_{\mathbf{p}} = no$ at the beginning of this phase, sends a message to all its neighbors in graph G_1 in the first round of this phase and waits for answer. If no answer contain $d = yes$, it sends message to all its neighbors in graph G_2, and so on. It is continued until receiving a decision value (that is, a message of the form $(1, 1)$ or $(0, 1)$). After receiving a decision value, it adopts it, becomes convinced and stops. Every convinced processor replies immediately after receiving a message from a non-convinced processor (in the same round). Every non-faulty processor $\mathbf{p}$ decides on its current value $v_{\mathbf{p}}$ at the end of round $\log n$ of this phase.

3.3.3 Correctness and complexity analysis

By construction, algorithm MSC satisfies termination condition. Clearly, each processor terminates by time $12t + 23 \log n + 23 \log n + 2 \log n = O(t + \log n)$.

Also, every processor may change its initial value only in the update after receiving messages, and the new value is received from some other processor. Therefore, by the inductive argument, in each round the set of current values v of processors is a subset of the set of all the initial values, and validity condition is satisfied.

We now prove that all correct processors decide on the same value.

Let B be the set of non-faulty processors at the end of the execution. Note that $|B| > 2n/3$. By the property of graph G_0, there exists a (2δ)-dense subset C of set B of size bigger than $\frac{n}{2}$ and diameter at most $23 \log n$ in the subgraph of G_0 induced by C.

Each node in C becomes convinced at the end of Phase 2, by the choice of set C. More precisely, this is because all nodes in set C keep receiving at least 2δ messages per round during the whole Phase 2. Consider the subgraph of G_0 induced by processors that are non-faulty by the end of Phase 1. If two processors are in the same connected component then they clearly have the same value at the end of Phase 1, and consequently by the end of Phase 2.

It follows from the following facts: (i) each time the value 1 is received for the first time by a processor, it is immediately re-sent to all its neighbors in G_0, and (ii) each path consisting of different nodes has length at most $n \leq 12t$. Indeed, a path from a processor **q** with initial value 1 that has been delivered to processor **p** can be extended to a path from **q** to any other node in the connected component of **p**. It can be done directly by concatenating the two sub-paths from **q** to **p** and from **p** to the other processor and using common points, if any, as shortcuts to avoid repetitions of nodes. Such path propagates value 1 and has length at most n, thus the other node in the connected component has value 1 by the end of Phase 1 as well.

It follows that at the end of Phase 2 all convinced processors have the same value: all within the same connected component as set C have the same values by the previous argument, while the number of processors outside this component is smaller than $\frac{n}{2}$, therefore by Lemma 3.2 none of them is in a 2δ-pseudo-dense, and therefore can not become convinced in Phase 2.

It remains to prove that every non-faulty processor becomes eventually convinced (by setting its variable d to *yes*). All processors in C are convinced by the end of Phase 2. We argue that by the end of round i of Phase 3, the number of unconvinced processors is smaller than $\frac{n}{2^i}$. The inductive proof is straightforward: in round $i+1$ graph G_{i+1} is used by unconvinced processors for sending requests. This graph is $\frac{n}{2^i}$-expanding graph, which means that there is no subset of size $\frac{n}{2^{i+1}}$ that would not be connected to the remaining processors (convinced and non-faulty). Thus at the end of Phase 3 all processors are convinced.

The above analysis of Phase 3 gives also the upper bound $O(n \log^2 n)$ for communication complexity. Indeed, in round i of this phase at most $\frac{n}{2^i}$ processors send $O(2^i \log n)$ messages each. Recall that each message consists of two bits, thus the number of communication bits is only twice as big as the number of messages sent. The number of responses is trivially bounded by the number of requests. Note that the communication incurred by Phases 1 and 2 is also $O(n \log^2 n)$, since every processor in graph G_0 sends $O(\delta)$ point-to-point messages at most $O(\log n)$ times. Thus we proved:

Lemma 3.3 *Algorithm MSC solves consensus in time $O(t)$ and with communication complexity $O(n \log^2 n)$, for at most t crashes, where $\frac{n}{12} \leq t < \frac{n}{3}$.*

Algorithm MSC can be easily extended to efficiently tolerate any $t < \frac{n}{3}$ crashes as follows. If $t < \log^2 n$ then we run a naive consensus algorithm for the first $t + 1$ processors, and then each of them spreads a decision to the remaining processors in one round. This works in time $O(t)$ and with communication complexity $O(nt) = O(n \log^2 n)$. For the case $\log^2 n \leq t < \frac{n}{12}$ we proceed as follows. The first two phases are performed only by the first $3t + 1$ processors, using graph G_0 designed for them (not for all processors!). Then

we run the third phase by all non-faulty processors. This phase differs from the restricted version only by the fact that before we use graphs $G_1, \ldots, G_{\log n}$ for sending request messages, we use a disperser graph H_t to guarantee that a large fraction of non-faulty processors (at least 2/3 of all non-faulty processors) become informed, and thus the remaining sub-phase involving graphs $G_1, \ldots G_{\log n}$ terminates successfully, as in the restricted case. Note that all graphs $H_t, G_1, \ldots, G_{\log n}$ are designed for *all* processors, unlike graph G_0 used in the first two phases. The asymptotic time complexity remains unchanged, that is $O(t)$, while the additional communication impact of disperser H is only $O(n \log n)$. Therefore we get the final result:

Theorem 3.4 *There exists a deterministic consensus algorithm working in time $O(t)$ and using $O(n \log^2 n)$ communication bits, for every $t < \frac{n}{3}$.*

3.4 Quantum algorithm

In this section we apply above communication scheme to the quantum model. We present an algorithm that solves consensus in time $O(\log n)$ using $O(n \log^3 n)$ qubits, and is $\frac{n}{3}$-crash-resilient.

We start with a simple formalization of the quantum model, consistent with our previous setting. Our model is equivalent to a standard quantum computation model, and uses notion of the Quantum Turing Machine. Again, we do not go deep into the details of this model. Reader interested in more broad view may found it in a book of Nielsen and Chuang [64].

3.4.1 Quantum distributed computation

To define properly model of quantum distributed computing, first we need to formalize a notion of a Quantum Turing Machine. Intuitively, this is a Turing Machine that may perform additional "quantum transitions". There are two types of them: *unitary transformation* and *measurement*.

State of the machine Since our consensus algorithm for n processors uses space of size $O(n^2)$, we here consider only machines with tapes bounded to this size. Therefore, for each n in fact we may consider any processor to be a finite state machine. Let a X denote the set of all possible states of such machine. Configuration of Quantum Turing Machine is a function $\Psi : \mathsf{X} \to \mathbb{C}$, where $\mathbb{C}$ is a set of complex numbers, called *amplitudes*. In order to be a proper state, this function must meet the requirement $\sum_{x \in \mathsf{X}} |\Psi(x)|^2 = 1$. If this function is equal 1 on one argument, and 0 on all the others, we say that machine is in classical configuration. If not, that is, this function is non-zero on more than one argument, we say that machine is in *superposition*.

We always assume that initial and final configuration of the machine is classical.

Unitary transformation Unitary transformation is a special type of transition allowed to the Quantum Turing Machine. In a classical model, we may present transitions of TM as a matrix with 1's and 0's, where each 1 denotes allowed step and 0 denotes a non-allowed step. In a deterministic TM, there is a single 1 in every column of this matrix. In such setting, computation is simply a repeated application of this matrix to a vector describing state of the machine.

If we present probabilistic machine in such a way, instead of 1's and 0's we may use other real positive numbers from the interval $[0, 1]$. The requirement is that values in every column must add to 1. Quantum model further extends this notion, allowing complex numbers instead of only real ones, and introducing two requirements. First is that $\forall_j \sum_i |M(i,j)|^2 = 1$. Second is that the matrix itself must be unitary ($M^*M = MM^* = I$).

The second requirement is not present in case of classical and probabilistic machines, and because of it, simulation of the classical computation on quantum machine is not trivial. It has been shown that it is possible. Easiest way to show that is to create a quantum simulation of every classical transition, storing on the tape all changes performed by the classical machine. Such operation is easy to reverse, and transition of such machines may all be unitary. This way, machine stores whole history of the simulated computation, and space complexity of the algorithm is of the size of time complexity of classical computation. There are some techniques that allow more efficient simulation. In this thesis we are not concerned by constant factors in costs, therefore the sole fact that such simulation is possible suffices.

Measurement Measurement reduces superposition to a singe classical state. Precisely, measurements is a transition that performed in state Ψ leads to each of the states $x \in \mathsf{X}$ with probability $|\Psi(x)|^2$. It must always lead to some state, and this is exactly the reason why these values must sum up to one. From the point of view of quantum theory, this is a complete measurement, which is a special case of more general *partial measurements*. The latter is more complicated and we will not be using it in our algorithm, therefore we omit it in this thesis.

It is worth noting, that even if in each state there is only a constant number of possible transitions, quantum machine may enter a superposition of exponential number of states in polynomial time. However, if we only need to compute the probability of one possible outcome with some bounded error, we may search the whole tree of states by depth first search algorithm and compute the probabilities on each branch with some polynomial accuracy.

This may be realized in $PSPACE$, which proves that quantum machines are simulated by alternating machines. The relation between quantum and nondeterministic machines is not determined for now.

Quantum distributed system Quantum distributed system Q is a set of Quantum Turing Machines with quantum communication interfaces. In this model, collective state of all machines and all tapes is described as one function Ψ. Special synchronization states and symbols are the same as in Section 1.2, and all previous notions apply. Only instead of a classical transition, each machine may use unitary transformations and measurements. In order to avoid the partial measurements, we use only simultaneous measurements of all machines. We assume that only correct machine may perform such measurement and it replaces quantum state of the whole system with a classical state.

It is possible to devise many nonequivalent models of quantum distributed system. Here we use one particularly realistic, in which quantum behavior is reserved only to memory qubits, not to behavior of the machines. It means that the number of qubits read and written by every machine in every round, number of rounds, destination of every message and its length - are all deterministic and known from the start. Only the content of the messages and tapes might be in superposition. Every instruction leaves heads of every machine in a classical state in one place (not in superposition), and at the end of each round every machine is in classical state $FinishRound$. Also, at the end of each round, before content of send buffers is moved to receive buffers, every receive buffer must be in a classical state (in particular, not entangled with any data on other tapes), to prevent non-unitary transformation performed by synchronization mechanism. This will be ensured in our algorithm.

Following the quantum mechanical Dirac notation, we shall use bracket to denote quantum states of the memory cells:

- $|x\rangle$ is a quantum state with amplitude 1 on classical state x, and 0 on all the others.
- $\alpha|x\rangle + \beta|y\rangle$ is a quantum state with amplitudes α on state x and β on state y, and so on.

We use the same notation for unitary transformations. Some of them, as for example quantum analogue of classical copying, will create entangled states of memory qubits. Term *entanglement* is a quantum analogue of dependent random variables in probabilistic settings. It means that state of two pieces of the system can not be described as a combination of two independent superpositions, but only by a singe function on all possible states of these pieces.

Quantum Turing Machine is inherently probabilistic and in some aspects similar to probabilistic Turing Machine. There are, nevertheless, two impor-

tant differences. First is that probabilities are real positive numbers, while amplitudes are complex ones. In consequence, amplitudes may subtract from each other, while probabilities only add up. Such process is called quantum interference and may be used to perform computations not possible in probabilistic model: e.g. Shor's algorithm [71] and Grover's algorithm [44]. Second difference, that is more important to us, is that transformations and measurements are separated in time. There is no additional "random tape" that may be read to perform the probabilistic transitions. Instead, Quantum Machine is literally in many states at the same time. To describe how this affects the model, we need to discuss one more notion about probabilistic and quantum algorithms: *the adversary*.

The Adversary Standard performance evaluation of an algorithm is to create so-called *worst-case scenario*, that is, to analyze it with an input for which it performs the worst. In distributed algorithms, this in particular encompasses the worst possible sequence of failures during the execution. When we were discussing deterministic algorithms, it was enough to propose a single such sequence for each of them, because the whole execution was then predetermined. This does not apply to probabilistic and quantum algorithms. If there is more than one possible behavior of the system, for each of them a different sequence of failures might be the worst. There are in fact several nonequivalent ways of defining the worst case scenario in this case. One efficient way of categorizing them is to introduce the adversary [3].

Adversary is a "second player", if we look at the execution as a game of solving the given problem. First player tries to fulfill the solution requirements, and his strategy is the algorithm itself. The adversary creates the input data and determines sequence of failures during the computation. His goal is to finish the execution incorrectly, and if it is impossible, to make the algorithm to use as many resources (time, messages, etc.) as possible. In a probabilistic model, there are three most common adversary types, defined by the amount of knowledge available to the adversary and the set of his possible strategies.

- **Oblivious adversary**
  Such adversary knows only the algorithm, and must prepare failure sequence before the execution starts.

- **Adaptive online adversary**
  Such adversary prepares the sequence during the execution, knowing everything that has happened in the system so far and all random bits used so far. E.g. it might crash each processor that received some number of messages, which the oblivious adversary can not (if this number is not predetermined).

- **Adaptive offline adversary**
  This adversary knows the random tapes/generators of every processor, and may predict all random values that will be used in the computation. In general, it looks at the probabilistic algorithm as a deterministic one.

These definitions induce three performance metrics for probabilistic algorithms, because in each case adversary may be able to force algorithm to work in different time and use different number of messages.

We may define similar classes of adversaries in a quantum model. The definitions of oblivious and adaptive online adversaries remains the same. Adaptive offline adversary is slightly different. There is no random tape known to him, instead the quantum machine is really in superposition of many states. Following the intuition that such adversary "knows everything", we say that he knows exact function Ψ at each moment of the execution. He can not, however, know the exact outcome of the measurement, because such thing has no meaning before the measurement is performed. In real world this would violate the uncertainty principle.

Now, we may see why it is important that in quantum model measurements are separated from the algorithm. Before the measurement is actually performed, even the all-knowing adversary can not predict its outcome. We may use it to strip the adversary from his ability to adapt, by postponing the measurements until the end of the execution.

Algorithm presented below works against the strongest, adaptive offline and all-knowing adversary, that may cause up to $\frac{n}{3}$ crashes during the execution.

Building blocks In this paragraph we present basic building blocks of quantum algorithm, in order to simplify its presentation. All transitions presented here are standard to quantum computation model or might be easily composed of standard quantum operation. To show that all of them are unitary, it is enough to show that all of them are involutions.

1. **Hadamard gate**
   This unitary transformation on a single qubit is described by matrix: $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
   It is worth noting that this gate applied to qubit in state $|0\rangle$ creates qubit in superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

2. **Superposition number**
   Application of Hadamard gate to every qubit in register $X = |00\ldots0\rangle$ (of k qubits), creates superposition of all values in this register:
   $H^{\otimes k}X = \frac{1}{2^{k/2}} \sum_{x=0}^{x=2^k-1} |x\rangle$ This is useful in generating superpositions and random numbers.

3. **CNOT(x,y)**
   CNOT (Controlled-NOT) is a basic operation on single qubits x and y, described by matrix:
   $$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$
   Let $x = \alpha|0\rangle + \beta|1\rangle$. The transition outcome is then:
   $CNOT(x, |y\rangle) = \alpha|0\rangle|y\rangle + \beta|1\rangle|1 \text{ xor } y\rangle$.

4. **Entangled_copy(A,B)**
   Let A and B be quantum registers of equal size, and B be initially set to $|00\ldots0\rangle$. Application of CNOT pairwise to corresponding qubits of A and B creates entangled copy of register A in place of B.
   Let $A = \sum_x \alpha_x|x\rangle$. The transition outcome is then $\sum_x \alpha_x|x\rangle|x\rangle$.

5. **Move(A,B)**
   Let A and B be a quantum registers of equal size, and B be initially set to $|00\ldots0\rangle$. Preparing an entangled copy of A on B, and then application of CNOT pairwise to corresponding qubits of B and A leaves A in state $|00\ldots0\rangle$, and B storing previous state of A.

6. **SWAP(A,B)**
   Let A and B be a quantum registers of equal size. SWAP operation exchanges content of those two registers.

7. **Conditional_NOT(*cond*, *input*, *output*)**
   Let *cond* be any boolean function on *input*, and *input* and *output* be quantum registers. It is possible to entangle *output* register with the value *con*(*input*) without measuring *input*. Let $input = \sum_x \alpha_x|x\rangle$. The transition outcome is then
   $Conditional\_NOT(cond, input, output) = \sum_x \alpha_x|x\rangle|cond(x) \text{ xor } output\rangle$.

8. **Conditional_SWAP(x, A, B)**
   Let x be one qubit register, and A and B be a quantum registers of equal size. As in previous case, it is possible to perform $SWAP(A, B)$ in an entangled way with register x, without measuring it. Let $x = \alpha|0\rangle + \beta|1\rangle$. The transition outcome is then
   $Conditional\_SWAP(x, A, B) = \alpha|0\rangle AB + \beta|1\rangle BA$
   If x was initially entangled with any other registers, state of registers A and B also become entangled with them.

3.4.2 Algorithm

The main technique used in this algorithm is sending messages in superpositions. If such message is entangled with some local data of the sender, state of the sender memory and receiver memory become entangled. In our algorithm we use this to create completely entangled state of the whole system before we make first measurement.

The following algorithm is a quantum version of randomized leader selection. Each processor selects a random *ticket number* and broadcasts it in the system, together with its consensus value. At the end, processor with the highest ticket number wins, and all processors adopt its consensus value. The range of values available for ticket numbers determines the probability of a collision. We take this range large enough to make this probability negligible.

It is easy to see, that against an oblivious adversary, that may use at most t crash failures, such algorithm works correctly with probability at least $1-\frac{t}{n}$. Adversary must choose which processors crash before ticket numbers are generated, therefore it has only $\frac{t}{n}$ chance to crash processor with the highest number. If not crashed, this processor will correctly broadcast its consensus value and all correct processors will adopt it.

The same is not true for adaptive adversaries. Consider the following strategy for adaptive adversary:

> Before the first message is sent, find a processor X with the highest ticket number, and processor Y with the highest ticket number among those with different consensus value than X. Crash all processors with ticket numbers between X and Y before they start sending messages and crash X in the middle of his broadcast, so that only a fraction of processors receive its messages.

Processors that receive message from X, will adopt its value, while the rest adopt value of Y - so the algorithm will end incorrectly. Observe that if there are similar numbers of initial 1's and 0's, this strategy requires only a constant expected number of crashes.

In a quantum model, there is a solution for this problem. Instead of choosing the ticket numbers and then broadcasting them, we may broadcast a superposition of all possible values, and measure it after messages are received. Algorithm presented below does exactly this, only in a slightly more refined way. Instead of simply broadcasting the values, we distribute them using a communication graph defined in the previous section. In each round, each processor receives values from its neighbors and sends the highest one to all of them. But since we do not want to measure these values, all comparisons between them are performed in a superpositions, using unitary transformations.

The pseudo-code of the algorithm for processor **p** is presented on Figure 3.1.

1. Prepare vote state on a single qubit: $|Vote_{\mathbf{p}}\rangle = |v_{\mathbf{p}}\rangle$
2. Take $3\log n$-qubit register $Ticket_{\mathbf{p}}$ and generate **Superposition number** on it: $|Ticket_{\mathbf{p}}\rangle = H^{3\log n}|00\ldots0\rangle = \frac{1}{2^{3/2\log n}}\sum_{a=1}^{2^{3\log n}}|a\rangle$
3. For $23\log n$ rounds:
   - For every neighbor **q** in G_0: Send an **Entangled copy** $|Ticket_{\mathbf{p}}\rangle|Vote_{\mathbf{p}}\rangle$ to **q**
   - **Move** every received message to fresh space in local memory. Update local memory in the following way:
     For each received message of the form $|Ticket_m\rangle|Vote_m\rangle$, prepare single qubit $S_m = |0\rangle$, and apply two unitary operations
     - **Conditional_NOT**$(>, (Ticket_m, Ticket_{\mathbf{p}}), S_m)$
     - **Conditional_SWAP**$(S_m, (Ticket_m, Vote_m), (Ticket_{\mathbf{p}}, Vote_{\mathbf{p}}))$
4. Measure local registers. Set $v_{\mathbf{p}}$ to outcome of the measurement of $Vote_{\mathbf{p}}$.
5. If in each round at least δ messages was received, set $d = yes$.
6. Perform *Phase 3* of the *Majority Sensing Consensus Algorithm.*

Figure 3.1: Quantum approximated consensus algorithm, pseudo-code for processor **p**

3.4.3 Analysis

Lemma 3.5 *If $t < \frac{n}{3}$, quantum approximated consensus algorithm correctly solves consensus with probability at least $\frac{1}{2}$*

Proof: Let us observe that by construction, destinations and length of all messages sent and received during the execution is deterministic and depends only on the sequence of failures. Only the content of these messages and local memory of the processors are in superposition. Consider now configuration of the whole system just after generation of the ticket numbers. This configuration has n^{3n} states with nonzero amplitude, corresponding to n^{3n} possible combinations of processor's ticket numbers. We may now analyze the execution of every of those combinations until the final measurement, and sum their impact on its result. The crucial point is that because there are no measurements before the final one, the adversary does not gain any information before this point. It means that sequence of failures must be the same for all

combinations.

In order to spoil the outcome, the adversary must ensure that consensus value of the processor with highest ticket number is not propagated correctly. Consider now any execution of the protocol with at most $\frac{n}{3}$ failures. Now we use the property of the graph G_0, that with any such sequence, there is a compact set C of at least $\frac{n}{2}$ processors. Consider now any combination of ticket numbers in which processor with highest number belongs to set C. Each time its ticked number is compared with another one, it wins and is propagated further. After $23 \log n$ rounds whole set C has its vote adopted as own. This set makes a decision and propagates it in the last line to all processors. It remains to compute what is the sum of modules of squares of amplitudes of all combinations where processor with the highest ticket number belongs to C. Every combination has the same amplitude $n^{-3n/2}$, therefore this sum is simply the ratio of $|C|$ to number of all processors, equal at least $\frac{1}{2}$.

□

Fact 1 *Quantum consensus algorithm works in time* $O(\log n)$ *and uses* $O(n \log^3 n)$ *qubits.*

Proof: By construction, algorithm works in time $O(\log n)$. In each round each processor sends $O(\log n)$ quantum messages, each of size $O(\log n)$ qubits, which sums up to $O(n \log^3 n)$ qubits.

□

By simple repetition of this algorithm, we can obtain consensus with arbitrarily high probability. If any iteration of this algorithm ends correctly, all further iteration starts with the same value in all processes, what guarantees their correctness.

Corollary 2 *For any positive* k*, there is a quantum algorithm that solves consensus with probability* $1 - 2^{-k}$ *in time* $O(k \log n)$ *and using* $O(kn \log^3 n)$ *qubits.*

3.5 Open problems

In this chapter we have shown a new, efficient way of communicate between processors in a distributed system, that allows them to perform consensus using very low number of bits. The "majority set" emerges from the system in a completely distributed way, without any processor knowing its form. Based solely on the properties of the underlying graph, we can however say enough about this set to prove correctness about the algorithm. This is very promising field of research of fully decentralized algorithms.

The main drawback of both algorithms presented here is that they work correctly only for a constant fraction of faulty processors. It would be interesting to improve this technique, using e.g. iterations similar to those used in gossip problem, to achieve resiliency for any number of failures.

Quantum algorithm has also the disadvantage of producing correct output only with some probability. Even though this probability may be chosen arbitrarily close to one, more desired property would be to finish always with a correct output. There are known techniques [35] to transform such Monte Carlo algorithm in to a Las Vegas one, such that it always ends correctly, at the expense of changing its execution time into a random value. These techniques, however, require communication overhead $\Omega(n^2)$, and therefore usage of them in our algorithm would undermine its efficiency. The possibility of obtaining sub-quadratic communication and correctness with probability 1 remains open.

There is also an open question of further improving the bit complexity of consensus algorithm. So far, the only known consensus algorithms that uses $O(n)$ communication bits are working in exponential time. It is a challenging problem to perform the same in polynomial time, in any model of computation: deterministic, randomized or a quantum one.

Bibliography

[1] Y. Afek and Y. De Levie, Space and step complexity of efficient adaptive collect, *Proc. of* 19*th International Symposium on Distributed Computing (DISC)*, 2005, pp. 384 - 398

[2] M.K. Aguilera, S. Toueg, A Simple Bivalency Proof that t-Resilient Consensus Requires t+1 Rounds, *Information Processing Letters*, 71 (1999) 155 - 158

[3] M. Ajtai, J. Aspnes, C. Dwork, O. Waarts, A theory of competetive analysis of distributed algorithms, *Proc. of* 35*th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994, pp. 401 - 411

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, *Computer Networks*, 38, 2002

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

[6] H. Attiya, F. Kuhn, C.G. Plaxton, M. Wattenhofer, and R. Wattenhofer, Efficient adaptive collect using randomization, *Distributed Computing* 18:3, 2006, pp. 179 - 188

[7] H. Attiya, J. Welch, Distributed Computing, John Willey & Sons, 2004 (second edition)

[8] Y. Aumann, M.A. Bender, Fault tolerant data structures, *Proc. of* 37*th Annual Symposium on Foundations of Computer Science (FOCS)*, 1996, pp. 580 - 589

[9] B. Awerbuch, Complexity of network synchronization, *Journal of ACM*, 32, 1985, pp. 804 - 823

[10] B. Baker, R. Shostak, Gossips and telephones, *Discrete Mathematics*, 2, 1972, pp. 191 - 193

[11] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, February 2003.

[12] R. Baldoni, J. Helary, M. Raynal, L. Tanguy, Consensus in Byzantine asynchronous systems. *Journal of Discrete Algorithms*, 1(2), 2003, pp. 185 - 210

[13] Z. Bar-Josephm, M. Ben-Or, A tight lower bound for randomized synchronous consensus, *Proc. of* 17*th ACM Symp. on Principles of Distributed Computing (PODC)*, 1998, pp. 193 - 199

[14] M. Ben-Or, Another advantage of free choice: Completely asynchronous consensus, *Proc. of* 2*nd ACM Symp. on Principles of Distributed Computing (PODC)*, 1983, pp. 27 - 30

[15] M. Ben-Or and A. Hassidim, Fast quantum byzantine agreement, *Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC)* 2005, pp. 481 - 485

[16] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM 13(7)*, 1970, pp. 422 - 426, doi:10.1145/362686.362692

[17] M.R. Capalbo, O. Reingold, S.P. Vadhan, and A. Wigderson, Randomness conductors and constant-degree lossless expanders, *Proc. of* 34*th Annual ACM Symposium on Theory of Computing (STOC)*, 2002, pp. 659 - 668

[18] B.S. Chlebus, R. De Prisco, A. Shvartsman, Performing tasks on restartable message-passing processors, *Distributed Computing* 14(1), 2001, pp. 49 - 64

[19] B.S. Chlebus, D.R. Kowalski, Robust gossiping with an application to consensus, *Journal of Computer and System Sciences*, 72, 2006, 1262 - 1281

[20] B.S. Chlebus, D. R. Kowalski, Time and communication efficient consensus for crash failures, *Proc. of* 20*th Symposium on Distributed Computing (DISC)*, 2006, LNCS 4167, pp. 314 - 328

[21] B.S. Chlebus, D. R. Kowalski, A. A. Shvartsman, Collective asynchronous reading with polylogarithmic worst-case overhead, *Proc. of* 36*th ACM Symposium on Theory of Computing (STOC)*, 2004, pp. 321 - 330

[22] M. Chrobak, L. Gasieniec, D. R. Kowalski, The wake-up problem in multi-hop radio networks, *Proc. of* 15*th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004, pp. 985 - 993.

[23] M. Chrobak, L. Gasieniec, W. Rytter, Fast broadcasting and gossiping in radio networks, *Journal of Algorithms*, 43(2) 2002, pp. 177 - 189

[24] B. N. Clark, C. J. Colbourn, D. S. Johnson, Unit disk graphs, *Discrete Mathematics*, 86, 1990, pp. 165 - 177

[25] M. Correia, P. Verssimo, N. F. Neves, Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey, *RESIST Network of Excellence*, 2006.

[26] F. Cristian, C. Fetzer, The timed asynchronous system model, *Proceedings of the* 28*th IEEE International Symposium on Fault-Tolerant Computing*, pp. 140 - 149

[27] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehear, D. Terry, Epidemic algorithms for replicated database maintenance, *Proc. of* 6*th ACM Symposium on Principles of Distributed Computing (PODC)*, 1987, pp. 1 - 12

[28] K. Diks and A. Pelc, Optimal Adaptive Broadcasting with a Bounded Fraction of Faulty Nodes, *Algorithmica* 28(1) (2000) 37-50

[29] D. Dolev and R. Reischuk, Bounds on information exchange for Byzantine Agreement, *Journal of ACM*, 32:1, 1985, pp. 191 - 204

[30] A. Doudou, B. Garbinato, R. Guerraoui, Encapsulating failure detection: From crash-stop to Byzantine failures, *International Conference on Reliable Software Technologies*, 2002, pp. 24 - 50

[31] A. Doudou, A. Schiper, Muteness detectors for consensus with Byzantine processes. *Technical Report 97/30, EPFL*, 1997.

[32] C. Dwork, J. Halpern, and O. Waarts, Performing work efficiently in the presence of faults, *SIAM Journal on Computing*, 27, 1998, pp. 1457 - 1491.

[33] C. Dwork, N. Lynch, and L. Stockmeyer, Consensus in the presence of partial synchrony, *Journal of ACM*, 35:2, 1988, pp. 288 - 323

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

[35] P. Feldman, S. Micali, An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement, *SIAM Journal of Computing*, 26 (4), 1997, pp. 873 - 933.

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

[37] M. Fisher, N. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, *A. ACM*, 32, 1985, pp. 374 - 382.

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

[39] J. Garay, Y. Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds, *SIAM Journal of Computing* 27(1), 1998.

[40] L. Gasieniec, T. Radzik, Q. Xin, Faster deterministic gossiping in directed ad-hoc radio networks, *Proc. of* 9*th Scandinavian Workshop on Algorithm Theory (SWAT)*, 2004, LNCS 3111, pp. 397 - 407

[41] C. Georgiou, D.R. Kowalski, and A.A. Shvartsman, Efficient gossip and robust distributed computation, *Proc. of* 17*th Int. Symposium on Distributed Computing (DISC)*, 2003, pp. 224 - 238

[42] C. Georgiou, A. Russel, A. A. Shvartsman, The complexity of synchronous iterative Do-All with crashes, *Distributed Computing*, 17(1), 2004, pp. 47 - 63

[43] S. Gilbert, R. Guerraoui, and D. Kowalski, On the message complexity of indulgent consensus, *Proc. of* 21*st International Symposium on Distributed Computing (DISC)*, 2007, pp. 283-297.

[44] L. K. Grover, A fast quantum mechanical algorithm for database search, *Proc. of 28th Annual ACM Symposium on the Theory of Computing (STOC)*, 1996, pp. 212-219

[45] V. Hadzilacos, Issues of Fault Tolerance in Concurrent Computations, PhD thesis, Harvard University, Cambridge, MA, 1984

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

[47] V. Hadzilacos, S. Toueg, Fault-tolerant broadcast and related problems, *Distributed Systems*, second edition, S. Mullender (Ed.), Eddison-Wesley, (1993), pp. 97 - 145.

[48] F. Harary, A. J. Schwenk, The Communication Problem on Graphs and Digraphs. *Journal Franklin Institute*, 297, 1974, pp. 491 - 495.

[49] M. Harchol-Balter, T. Leighton, D. Lewin, Resource discovery in distributed networks, *Proc. of* 18*th ACM Symposium on Principles of Distributed Computing (PODC)*, 1999, pp. 229 - 238

[50] S. M. Hedetniemi, S. T. Hedetniemi, A. L. Liestman, A Survey of Gossiping and Broadcasting in Communication Networks, *Networks* 18, 1988, pp. 319 - 349.

[51] S. Hoory, N. Linial, A. Wigderson, Expander graphs and their applications, *Bulletin of the American Mathematica Society*, 43 (4), 2006, pp. 439 561 S 0273-0979(06)01126-8

[52] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzicka, and W. Unger, Dissemination of information in communication networks: broadcasting, gossiping, leader election, and fault-tolerance, Theoretical Computer Science, EATCS Series, Springer, 2005

[53] R. M. Karp, C. Schindelhauer, S. Shenker, B. Voecking, Randomized rumor spreading, *Proc.* 41*st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pp. 565 - 574

[54] D. Kempe, J. Kleinberg, A. Demers, Spatial gossip and resource location protocols, *Journal of ACM*, 51/6, 2004, pp. 943 - 967

[55] D. R. Kowalski, M. Strojnowski, On the Communication Surplus Incurred by Faulty Processors, *Proc. of* 21*th Symposium on Distributed Computing (DISC)*, 2007, LNCS 4731, pp. 328 - 342

[56] L. Lamport, Paxos Made Simple, em ACM SIGACT News (Distributed Computing Column) 2001, pp. 51-58.

[57] L. Lamport, Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM 21 (7)*, 1978, pp. 558 565. doi:10.1145/359545.359563

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

[59] M. Li, P. Vitanyi, Reversible Simulation of Irreversible Computation, *11th Annual IEEE Conference on Computational Complexity (CCC'96)*, p. 301

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

[61] N. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers, 1996

[62] N. A. Lynch, R. Segala, F. W. Vaandrager, Hybrid I/O automata, "Centrum voor Wiskunde en Informatica (CWI)",

[63] G. Neiger and S. Toueg, Automatically increasing the fault-tolerance of distributed systems, *Journal of Algorithms*, 11, 1990, pp. 374 - 419

[64] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000

[65] M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, *Journal of ACM*, 27, 1980, pp. 228 - 234.

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

[67] K. J. Perry, S. Toueg, Distributed agreement in the presence of processor and communication faults, *IEEE Transactions on Software Engineering*, 12(3), 1986, pp. 477 - 482

[68] R. van Renesse, Y. Minsky, M. Hayden, A gossip-style failure detection service, in *Proc. of Middleware*, 1998, pp. 55 - 70

[69] M. Saks, N. Shavit, H. Woll, Optimal time randomized consensus - making resilient algorithms fast in practice, *Proc. of 2nd SIAM-ACM Symposium on Discrete Algorithms (SODA)*, 1991, pp. 351 - 362.

[70] R.D. Schlichting, F.B. Schneider, Fail-stop processors: An approach to designing fault-tolerant computing systems, *ACM Transactions on Computing Systems*, 1:3, 1983, pp. 222 - 238

[71] P. W. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, *Journal of Sci.Stat. Computations (SIAM)*, 26, 1997

[72] W. T. Sullivan et. al. A new major SETI project based on Project Serendip data and 100,000 personal computers, *Proc. of the Fifth International Conference on Bioastronomy* http://seticlassic.ssl.berkeley.edu/woody_paper.html

[73] E. Upfal, Tolerating a linear number of faults in networks of bounded degree, *Information and Computation*, 115:2, 1994, pp. 312 - 320

[74] A. Ta-Shma, C. Umans, and D. Zuckerman, Lossless condensers, unbalanced expanders, and extractors, *Combinatorica*, 27, 2007, pp. 213 - 240.

[75] R. Tijdeman, On a Telephone Problem, *Nieuw Archief voor Wiskunde*, 19, 1971, pp. 188 - 192.

[76] S. Voulgaris, M. van Steen, An epidemic protocol for managing routing tables in very large peer-to-peer networks, *Proc. of Distributed Sys., Operations and Management (DSOM)*, 2003, pp. 41 - 54

[77] B.Y. Zhao, H. Ling, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment *IEEE Journal onSelected Areas in Communications*, Volume 22, Issue 1, 2004, pp. 41 - 53

Index