Improved Communication Complexity of Fault-Tolerant Consensus

Mohammad T. Hajiaghayi
University of Maryland
USA

Dariusz R. Kowalski
Augusta University, Georgia
USA

Jan Olkowski
University of Maryland
USA

ABSTRACT

Consensus is one of the most thoroughly studied problems in distributed computing, yet there are still complexity gaps that have not been bridged for decades. In particular, in the classical message-passing setting with processes' crashes, since the seminal works of Bar-Joseph and Ben-Or [PODC 1998] and Aspnes and Waarts [SICOMP 1996, JACM 1998] in the previous century, there is still a fundamental unresolved question about communication complexity of fast randomized Consensus against a (strong) adaptive adversary crashing processes arbitrarily online. The best known upper bound on the number of communication bits is $\Theta(\frac{n^{3/2}}{\sqrt{\log n}})$ per process, while the best lower bound is $\Omega(1)$. This is in contrast to randomized Consensus against a (weak) oblivious adversary, for which time-almost-optimal algorithms guarantee amortized $O(1)$ communication bits per process. We design an algorithm against adaptive adversary that reduces the communication gap by nearly linear factor to $O(\sqrt{n} \cdot \text{polylog } n)$ bits per process, while keeping almost-optimal (up to factor $O(\log^3 n)$) time complexity $O(\sqrt{n} \cdot \log^{5/2} n)$.

More surprisingly, we show this complexity indeed can be lowered further, but at the expense of increasing time complexity, i.e., there is a *trade-off* between communication complexity and time complexity. More specifically, our main Consensus algorithm allows to reduce communication complexity per process to any value from polylog n to $O(\sqrt{n} \cdot \text{polylog } n)$, as long as Time $\times$ Communication $= O(n \cdot \text{polylog } n)$. Similarly, reducing time complexity requires more random bits per process, i.e., Time $\times$ Randomness $= O(n \cdot \text{polylog } n)$.

Our parameterized consensus solutions are based on a few newly developed paradigms and algorithms for crash-resilient computing, interesting on their own. The first one, called a *Fuzzy Counting*, provides for each process a number which is in-between the numbers of alive processes at the end and in the beginning of the counting. Our deterministic Fuzzy Counting algorithm works in $O(\log^3 n)$ rounds and uses only $O(\text{polylog } n)$ amortized communication bits per process, unlike previous solutions to counting that required $\Omega(n)$ bits. This improvement is possible due to a new *Fault-tolerant Gossip* solution with $O(\log^3 n)$ rounds using only $O(|\mathcal{R}| \cdot \text{polylog } n)$ communication bits per process, where $|\mathcal{R}|$ is the length of the rumor binary representation. It exploits distributed fault-tolerant divide-and-conquer idea, in which processes run a *Bipartite Gossip* algorithm for a considered partition of processes. To avoid passing many long messages, processes use a family of small-degree compact expanders for *local signaling* to their overlay neighbors if they are in a compact (large and well-connected) party, and switch to a denser overlay graph whenever local signalling in the current one is failed.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**.

KEYWORDS

distributed consensus, crash failures, adaptive adversary

ACM Reference Format:
Mohammad T. Hajiaghayi, Dariusz R. Kowalski, and Jan Olkowski. 2022. Improved Communication Complexity of Fault-Tolerant Consensus. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3520078

1 INTRODUCTION

Fault-tolerant Consensus – when a number of autonomous processes want to agree on a common value among the initial ones, despite of failures of processes or communication medium – is among foundation problems in distributed computing. Since its introduction by Pease, Shostak and Lamport [31], a large number of algorithms and impossibility results have been developed and analyzed, applied to solve other problems in distributed computing and systems, and led to a discovery of a number of new important problems and solutions, c.f., [8]. Despite this persistent effort, we are still far from obtaining even asymptotically optimal solutions in most of the classical distributed models.

In particular, in the classical message-passing setting with processes' crashes, despite of the results obtained in the seminal works of Bar-Joseph and Ben-Or [9] and Aspnes and Waarts [5, 6] in the previous century, there is still a substantial gap in communication complexity of fast randomized Consensus. More precisely, in this model, n processes communicate and compute in synchronous rounds, by sending/receiving messages to/from a subset of processes and performing local computation. Each process knows set $\mathcal{P}$ of IDs of all n processes. Up to $f < n$ processes may crash accidentally during the computation, which is typically modeled by an abstract adversary that selects which processes to crash and when, and additionally – which messages sent by the crashed processes could reach successfully their destinations. An execution of an algorithm against an adversary could be seen as a game, in which the algorithm wants to minimize its complexity measures (such as time and communication bits) while the adversary aims at violating this goal by crashing participating processes. The classical

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3520078

distributed computing focuses on two main types of the adversary: adaptive and oblivious. Both of them know the algorithm in advance, however the former is stronger as it can observe the run of the algorithm and decide on crashes online, while the latter has to fix the schedule of crashes in advance (before the algorithm starts its run). Thus, these adversaries have different power against randomized algorithms, but same against deterministic ones.

One of the perturbations caused by crashes is that they substantially delay reaching consensus: no deterministic algorithm can reach consensus in all admissible executions within f rounds, as proved by Fisher and Lynch [18], and no randomized solution can do it in $o(\sqrt{n/\log n})$ expected number of rounds against an adaptive adversary, as proved by Bar-Joseph and Ben-Or [9]. Both these results have been proven (asymptotically) optimal. The situation gets more complicated if one seeks *time-and-communication* optimal solutions. The only existing lower bound requires $\Omega(n)$ messages to be sent by any algorithm even in some failure-free executions, which gives $\Omega(1)$ bits per process [4].* which is the total communication complexity divided by n. There exists even a *deterministic* algorithm with a polylogarithmic amortized number of communication bits [14], however it requires a linear number of rounds (as any deterministic solution, see [18]). On the other hand, randomized algorithms running against *weak adversaries* are both fast and amortized-communication-efficient, both formulas being $O(\log n)$ or better, c.f., Gilbert and Kowalski [22]. At the same time, the best randomized solutions against an *adaptive adversary* considered in this work requires time $\Theta(\sqrt{n/\log n})$ but large amortized communication $\Theta(n \cdot \sqrt{n/\log n})$. In this paper, we show a parameterized algorithm not only improves amortized communication by nearly a linear factor, but also suggests surprisingly that there is no time-and-communication optimal algorithm in this setting. The omitted analysis' details, due to space limit, could be found in [26].

Consensus problem. Consensus is about making a common decision on some of the processes' input values by every non-crashed process, and is specified by the three requirements:

Validity: Only one of the initial values may be decided upon.
Agreement: No two processes decide on different values.
Termination: Each alive process eventually decides.

All the above requirements must hold with probability 1. We focus on *binary consensus*, in which initial values are in $\{0, 1\}$.

2 OUR RESULTS AND NEW TOOLS

Our main result is a new consensus algorithm ParameterizedConsensus*, parameterized by x, that achieves *any* asymptotic time complexity between $\tilde{O}(\sqrt{n})$† and $\tilde{O}(n)$, while preserving the consensus complexity equation: Time × Amortized_Communication = $O(n \text{ polylog } n)$. This is also the first algorithm that makes a smooth transition between a class of algorithms with the optimal running time (c.f., Bar-Joseph's and Ben-Or's [9] randomized algorithm that works in $\tilde{O}(\sqrt{n})$ rounds) and the class of algorithms with amortized polylogarithmic communication bit complexity (c.f., Chlebus, Kowalski and Strojnowski [14] deterministic algorithm using $\tilde{O}(1)$ communication bits).

Theorem 1 (Section 5.4). *For any $x \in [1, n]$ and the number of crashes $f < n$, ParameterizedConsensus* solves Consensus with probability 1, in $\tilde{O}(\sqrt{nx})$ time and $\tilde{O}(\sqrt{n/x})$ amortized bit communication complexity, whp, using $\tilde{O}(\sqrt{n/x})$ random bits per process.*

In this section we only give an overview of the most novel and challenging part of ParameterizedConsensus*, called ParameterizedConsensus, which solves Consensus if the number of failures $f < \frac{n}{10}$. Its generalization to ParameterizedConsensus* is done in Section 5.4, by exploiting the concept of epochs in a similar way to [9, 14]. In short, the first and main epoch (in our case, ParameterizedConsensus followed by BiasedConsensus described in Section 2.1) is repeated $O(\log n)$ times, each time adjusting expansion, density and probability parameters by factor equal to $\frac{9}{10}$. The complexities of the resulting algorithm are multiplied by a logarithmic factor.

High-level idea of ParameterizedConsensus**.** In ParameterizedConsensus, processes are clustered into x disjoint groups, called super-processes $SP_1, \ldots, SP_x$, of $\frac{n}{x}$ processes each. Each process, in a local computation, initiates its candidate value to the initial value, pre-computes the super-process it belongs to, as well as two expander-like overlay graphs which are later use to communicate with other processes.

Degree δ of both overlay graphs is $O(\log n)$, and correspondingly the edge density, expansion and compactness are selected, c.f., Sections 4 and 5. One overlay graph, denoted $\mathcal{H}$, is spanned on the set of x super-processes, while copies of the other overlay graph are spanned on the members of each pair of super-processes SP_i, SP_j connected by an edge in $\mathcal{H}$ (we denote such copy by $SE(SP_i, SP_j)$).

ParameterizedConsensus is split into three phases, c.f., Algorithm 7 in Section 5. Each phase uses some of the newly developed tools, described later in this section: α-BiasedConsensus and Gossip. Processes keep modifying their candidate values, starting from the initial values, through different interactions.

Using the tools. α-BiasedConsensus is used for maintaining the same candidate value within each super-process, biasing it towards 0 if less than a certain fraction α of members prefer 1; see description in Section 2.1 and 6. Theorem 2 proves that α-BiasedConsensus works correctly in $\tilde{O}(\sqrt{n/x})$ time and communication bits per process. Gossip, on the other hand, is used to propagate values between all or a specified group of processes, see description in Section 2.2 and 7.2. Theorem 3 guarantees that Gossip allows to exchange information between the involved up to n' processes, where $n' \leq n$, in time $O(\log^3 n)$ and using $O(\log^6 n)$ communication bits per process (in this application, we are using a constant number of rumors, encoded by constant number of bits).

In Phase 1, super-processes want to flood value 1 along an overlay graph $\mathcal{H}$ of super-processes, to make sure that processes in the same connected component of $\mathcal{H}$ have the same candidate value at the end of Phase 1. Here by a connected component of graph $\mathcal{H}$ we understand a maximum connected sub-graph of $\mathcal{H}$ induced by super-processes of at least $\frac{3}{4} \cdot \frac{n}{x}$ non-faulty processes; we call such super-processes non-faulty. Recall, that the adversary can disconnect super-processes in $\mathcal{H}$ by crashing some members of

*In this paper we typically state communication complexity results in terms of *amortized per process*,
†We use $\tilde{O}$ symbol to hide any polylog n factors.

selected super-processes. To do so, the following is repeated $x+1$ times: processes in a non-faulty super-process SP_i, upon receiving value 1 from some neighboring non-faulty super-process, make agreement (using BiasedConsensus) to set up their candidate value to 1 and send it to all their neighboring super-processes SP_j via links in overlay graphs $SE(SP_i, SP_j)$. One of the challenges that need to be overcome is inconsistency in receiving value 1 by members of the same super-process, as – due to crashes – only some of them may receive the value while others may not. We will show that it is enough to assume threshold $\frac{2}{3}$ in the BiasedConsensus, which together with expansion of overlay graphs $SE(SP_i, SP_j)$ and compactness of $\mathcal{H}$ (existence of large sub-component with small diameter, c.f., Lemma 2) guarantee propagation of value 1 across the whole connected component in $\mathcal{H}$. It all takes $(x+1)\cdot(\tilde{O}(\sqrt{n/x})+1) = \tilde{O}(\sqrt{xn})$ rounds and $\tilde{O}(\sqrt{n/x}+\log n) = \tilde{O}(\sqrt{n/x})$ amortized communication per process; see Section 5.1 for details.

In Phase 2, non-faulty super-processes want to estimate the number of non-faulty super-processes in the neighborhood of radius $O(\log x)$ in graph $\mathcal{H}$. (We know from Phase 1 that whole connected non-faulty component in $\mathcal{H}$ has the same candidate value.) In order to do it, they become "active" and keep exchanging candidate value 1 with their neighboring super-processes in overlay graph $\mathcal{H}$ in stages, until the number of "active" neighbors becomes less or equal to a threshold $\delta_x = \Theta(\log x) < \delta$, in which case the super-process becomes inactive, but not more than than $\gamma_x = O(\log x)$ stages. To assure proper message exchange between neighboring super-processes, Gossip is employed on the union of members of every neighboring pair of super-processes. It is followed by BiasedConsensus within each active super-process to let all its members agree if the threshold δ_x on the number of active neighbors holds. Members of those super-processes who stayed active by the end of stage γ_x ("survived") conclude that there was at least a certain constant fraction of non-faulty super-processes (each containing at least a fraction of non-faulty members) in such neighborhood in the beginning of Phase 2, and thus they set up variable `confirmed` to 1 – it means they confirmed being in sufficiently large group having the same candidate value and thus they are entitled to decide and make the whole system to decide on their candidate value. It all takes $\gamma_x \cdot \tilde{O}(\sqrt{n/x}+\log^3 n) \le \tilde{O}(\sqrt{xn})$ rounds and at most $\gamma_x \cdot \delta \cdot \tilde{O}(\log^6 n + \sqrt{n/x}) = \tilde{O}(\sqrt{n/x})$ amortized communication per process. See Section 5.3 for further details.

In Phase 3, we discard the partition into x super-processes. All processes want to learn if there was a sufficiently large group confirming the same candidate value in Phase 2. To do so, they all execute the Gossip algorithm. Processes that set up variable `confirmed` to 1 start the Gossip algorithm with their rumor being their candidate value; other processes start with a null value. Because super-processes use graph $\mathcal{H}$ for communication, which in particular satisfies $(\frac{x}{64}, \frac{3}{4}, \delta_x)$-compactness property (i.e., from any subset of at least $\frac{x}{64}$ super nodes one can choose at least $\frac{3}{4}$ of them such that they induced a subgraph of degree at least δ_x), we will prove that at the end of Phase 2 at least a constant fraction of super-processes must have survived and be non-faulty (i.e., their constant fraction of members is alive). Moreover, we show that there could be only one non-faulty connected component of confirmed processes, by expansion of graph $\mathcal{H}$ that would connect two components of constant fraction of super-processes each (and thus would have propagated value 1 from one of them to another in Phase 1) – hence, there could be only one non-null rumor in the Gossip, originated in a constant fraction of processes. By property of Gossip, each non-faulty process gets the rumor and decides on it. It all takes $\tilde{O}(\log^3 n) \le \tilde{O}(\sqrt{xn})$ rounds and at most $\tilde{O}(\log^6 n) = \tilde{O}(\sqrt{n/x})$ amortized communication per process; see Section 5.2 for details.

Summarizing, each part takes $\tilde{O}(\sqrt{xn})$ rounds and $\tilde{O}(\sqrt{n/x})$ amortized communication per process. Each process uses random bits only in executions of BiasedConsensus it is involved to, each requiring $\tilde{O}(\sqrt{n/x})$ random bits (at most one random bit per round). The number of such executions is $O(x)$ in Part 1 and $O(\log n)$ in Part 2, which in total gives $\tilde{O}(\sqrt{xn})$ random bits per process.

2.1 α-Biased Consensus

Let us start with the formal definition of α-Biased Consensus.

Definition 1 (α-Biased Consensus). *An algorithm solves α-Biased Consensus if it solves the Consensus problem and additionally, the consensus value is* 0 *if less than αn initial values of processes are* 1.

In Section 6, we design an efficient α-Biased Consensus algorithm and prove the following:

Theorem 2 (Section 6). *For every constant $\alpha > 0$, there exists an algorithm, called α-BiasedConsensus, that solves α-Biased Consensus problem with probability* 1*, in $\tilde{O}(f/\sqrt{n})$ rounds and using $\tilde{O}(f/\sqrt{n})$ amortized communication bits whp, for any number of crashes $f < n$.*

Note that for $f = \Theta(n)$ the algorithm works in $\tilde{O}(\sqrt{n})$ rounds and uses $\tilde{O}(\sqrt{n})$ communication bits per process. Observe also that the above result solves classic Consensus as well, and as a such, it is the first algorithm which improves on the amortized communication of Bar-Joseph's and Ben-Or's Consensus algorithm [9], which has been known as the best result up for over 20 years. The improvement is by a nearly linear factor $\Theta(n/\log^{13/2} n)$, while being only $O(\log^3 n)$ away from the absolute lower bound on time complexity (also proved in [9]).

High-level idea of α-BiasedConsensus**.** The improvement comes from replacing a direct communication, in which originally all processes were exchanging their candidate values, by procedure FuzzyCounting. This deterministic procedure solves Fuzzy Counting problem, i.e., each process outputs a number between the starting and ending number of active processes, and does it in $O(\log^3 n)$ rounds and with $O(\log^7 n)$ communication bits per process, see Sections 2.3, 7 and Theorem 4.

First, processes run FuzzyCounting where the set of active processes consists of the processes with input value 1. Then, each process calculates logical *AND* of the two values: its initial value and the logical value of formula "$\texttt{ones} \ge \alpha \cdot n$", where `ones` is the number of 1's output by the FuzzyCounting algorithm. Denote by x_p the output of the logical *AND* calculated by process p – it becomes p's candidate value.

Next, processes run $O(f/\sqrt{n\log n})$ *phases* to update their candidate values such that eventually every process keeps the same choice. To do so, in a round r every process p calculates, using the FuzzyCounting algorithm, the number of processes with (current)

candidate value 1 and, separately, the number of processes with (current) candidate value 0, denoted O_p^r and Z_p^r respectively. Based on these numbers, process p either sets its candidate value to 1, if the number O_p^r is large enough, or it sets it to 0, if the number Z_p^r is large, or it replaces it with a random bit, if the number of zeros and ones are close to each other.

In Bar-Joseph's and Ben-Or's algorithm the numbers Z_p^r and O_p^r were calculated in a single round of all-to-all communication. However, we observe that because processes' crashes may affect this calculation process in an arbitrary way (the adversary could decide which messages of the recently crashed processes to deliver and which do not, see Section 4) and also because messages are simply zeros and ones, this step can be replaced by any solution to Fuzzy Counting. In particular, the correctness and time complexity analysis of the original Bar-Joseph's and Ben-Or's algorithm captured the case when an arbitrary subset of 0-1 messages from processes alive in the beginning of this step and a superset of those alive at the end of the step could be received and counted – and this can be done by our solution to the Fuzzy Counting problem.

Monte Carlo version for $f = n - 1$. α-BiasedConsensus as described above is a Las Vegas algorithm with an expected time complexity $\tau = \tilde{O}(\sqrt{n})$, as is the original Bar-Joseph's and Ben-Or's algorithm on which it builds. However, we can make it Monte Carlo, which is more suitable for application in ParameterizedConsensus, by forcing all processes to stop by time $const \cdot \tau$. In such case, the worst-case running time will always be while the correctness (agreement) will hold only whp. In order to be applied as a subroutine in the ParameterizedConsensus, we need to add one more adjustment, so that ParameterizedConsensus could guarantee correctness with probability 1. Mainly, processes which do not decide by time $const \cdot \tau - 2$ initiate a 2-round switch of the *whole system of $\mathcal{P}$ processes* to a deterministic consensus algorithm, that finishes in $O(n)$ rounds and uses $O(\text{polylog } n)$ communication bits per process, e.g., from [14]. Such switch between two consensus algorithms has already been designed and analyzed before, c.f., [14], and since this scenario happens only with polynomially small probability, the final time complexity of ParameterizedConsensus will be still $\tilde{O}(\sqrt{xn})$ and bit complexity $\tilde{O}(\sqrt{n/x})$ per process, both whp and expected.

2.2 Improved Fault-Tolerant Gossip Solution

The ParameterizedConsensus algorithm relies on a new (deterministic) solution to a well-known Fault-Tolerant Gossip problem, in which each non-faulty process has to learn initial rumors of all other non-faulty processes (while it could or could not learn some initial rumors of processes that crash during the execution). Many solutions to this problem have been proposed (c.f., [3, 10]), yet, the best deterministic algorithm given in [10] solves Fault-tolerant Gossip in $O(\log^3 n)$ rounds using $O(\log^4 n)$ point-to-point messages amortized per process. However, it requires $\Omega(n)$ amortized communication bits regardless of the size of rumors. We improve this result as follows:

Theorem 3 (Section 7.2). *Gossip solves deterministically the Fault-tolerant Gossip problem in $\tilde{O}(1)$ rounds using $\tilde{O}(|\mathcal{R}|)$ amortized number of communication bits, where $|\mathcal{R}|$ is the number of bits needed to encode the rumors.*

High-level idea of Gossip**.** The algorithm implements a distributed divide-and-conquer approach that utilizes the BipartiteGossip deterministic algorithm, described in Section 2.4, in the recursive calls. Each process takes the set $\mathcal{P}$, an initial rumor r and its unique name $p \in \mathcal{P}$ as an input. The processes split themselves into two groups of size at most $\lceil n/2 \rceil$ each: the first $\lceil n/2 \rceil$ processes with the smallest names make the group $\mathcal{P}_1$, while the $n - \lceil n/2 \rceil$ processes with the largest names constitute group $\mathcal{P}_2$. Each of those two groups of processes solves Gossip separately, by evoking the Gossip algorithm inside the group only. The processes from each group know the names of every other process in that group, hence the necessary conditions to execute the Gossip recursively are satisfied. After the recursion finishes, a process in $\mathcal{P}_1$ stores a set of rumors $\mathcal{R}_1$ of processes from its group, and respectively, a process in $\mathcal{P}_2$ stores a set of rumors $\mathcal{R}_2$ of processes from its group. Then, the processes solve the Bipartite Gossip problem by executing the BipartiteGossip algorithm on the partition $\mathcal{P}_1, \mathcal{P}_2$ and having initial rumors $\mathcal{R}_1$ and $\mathcal{R}_2$. The output of this algorithm is the final output of the Gossip. A standard inductive analysis of recursion and Theorem 5 stating correctness and $\tilde{O}(1)$ time and $\tilde{O}(|\mathcal{R}|)$ amortized communication complexities of BipartiteGossip imply Theorem 3, which proof is deferred to Section 7.2.

2.3 Fuzzy Counting

The aforementioned improvement of algorithm α-BiasedConsensus over [9] is possible because of designing and employing an efficient solution to a newly introduced Fuzzy Counting problem.

Definition 2 (Fuzzy Counting). *An algorithm solves Fuzzy Counting if each process returns a number between the initial and the final number of active processes. Here, being active depends on the goal of the counting, e.g., all non-faulty processes, processes with initial value* 1*, etc.*

Note that the returned numbers could be different across processes. In Section 7 we design a deterministic algorithm FuzzyCounting and prove the following:

Theorem 4 (Section 7.2). *The FuzzyCounting deterministic algorithm solves the Fuzzy Counting problem in $\tilde{O}(1)$ rounds, using $\tilde{O}(1)$ communication bits amortized per process.*

High-level idea of FuzzyCounting**.** FuzzyCounting uses the Gossip algorithm with the only modification that now we require the algorithm the return the values $\mathtt{Z}$ and $\mathtt{0}$, instead of the set of learned rumors. We apply the same divide-and-conquer approach. That is, we partition $\mathcal{P}$ into groups $\mathcal{P}_1$ and $\mathcal{P}_2$ and we solve the problem within processors of this partition. Let $\mathtt{Z}_1$, $\mathtt{0}_1$ and $\mathtt{Z}_2$, $\mathtt{0}_2$ be the values returned by recursive calls on set of processes $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Then, we use the BipartiteGossip algorithm, described in Section 2.4, to make each process learn values $\mathtt{Z}$ and $\mathtt{0}$ of the other group. Eventually, a process returns a pair of values $\mathtt{Z}_1 + \mathtt{Z}_2$ and $\mathtt{0}_1 + \mathtt{0}_2$ if it received the values from the other partition during the execution of BipartiteGossip, or it returns the values corresponding to the recursive call in its own partition otherwise. It is easy to observe that during this modified execution processes

must carry messages that are able to encode values Z and 0, thus in this have it holds that $|\mathcal{R}| = O(\log n)$.

2.4 Bipartite Gossip

Our Gossip and FuzzyCounting algorithms use subroutine BipartiteGossip that solves the following (newly introduced) problem.

Definition 3. *Assume that there are only two different rumors present in the system, each in at most $\lceil \frac{n}{2} \rceil$ processes. The partition is known to each process, but the rumor in the other part is not. We say that an algorithm solves Bipartite Gossip if every non-faulty process learns all rumors of other non-faulty processes in this setting.*

Bipartite Gossip is a restricted version of the general *Fault-tolerant Gossip* problem, which can be solved in $O(\log^3 n)$ rounds using $O(\log^4 n)$ point-to-point messages amortized per process, but requires $\Omega(n)$ amortized communication bits. In this paper, we give a new efficient deterministic solution to Bipartite Gossip, called BipartiteGossip, which, properly utilized, leads to better solutions to Fault-tolerant Gossip and Fuzzy Counting. More details and the proof of the following result are given in Section 7.1.

Theorem 5 (Section 7.1). *Given a partition of the set of processes $\mathcal{P}$ into two groups $\mathcal{P}_1$ and $\mathcal{P}_2$ of size at most $\lceil n/2 \rceil$ each, deterministic algorithm* BipartiteGossip *solves the Bipartite Gossip problem in $\tilde{O}(1)$ rounds and uses $\tilde{O}(n \cdot |\mathcal{R}|)$ bits, where $|\mathcal{R}|$ is the minimal number of bits needed to uniquely encode the two rumors.*

High-level idea of BipartiteGossip**.** If there were no crashes in the system, it would be enough if processes span a bipartite expanding graph with poly-logarithmic degree on the set of vertices $\mathcal{P}_1 \cup \mathcal{P}_2$ and exchange messages with their initial rumors in $\tilde{O}(1)$ rounds. In this ideal scenario the $O(\log n)$ bound on the expander diameter suffices to allow every two process exchange information, while the sparse nature of the expander graphs contributes to the low communication bit complexity. However, a malicious crash pattern can easily disturb such a naive approach. To overcome this, in our algorithm processes – rather than communicating exclusively with the other side of the partition – also estimate the number of crashes in their own group. Based on its result, they are able to adapt the level of expansion of the bipartite graph between the two parts to the actual number of crashes. More specifically, in *internal communication* within each group, a family of certain expander graphs (c.f., Theorem 6) with different density is adaptively and locally used by processes to exchange messages. Once a process recognizes (via Local Signalling, c.f., Section 2.5) that it does not belong to a large and compact component, it switches to a denser expander. In *external communication*, processes use a different family of expanders of different densities to communicate with processes in the other group in order to get their rumor – the degree of the chosen expander depends on current degree used in the internal communication.

The above dynamic adjustment of internal and external communication degree allows to achieve asymptotically similar result as in the fault-free scenario described in the beginning, up to polylogarithmic factor. More details and the analysis are in Section 7.1.

2.5 Local Signalling

Our BipartiteGossip algorithm, described in section 2.4, uses a new technique called LocalSignalling. LocalSignalling is a specific deterministic algorithm, parameterized by a family of $O(\log n)$ overlay graphs (of different density) provided to the processes. Processes start at the same time, but may be at different levels – the level indicates which overlay graph is used for communication. The name Local Signalling comes from the way it works – similarly to distributed sparking networks, a process keeps sending messages (i.e., 'signalling') to its neighbors in its current overlay graph as long as it receives enough number of messages from them. Once a process fails to receive a sufficient number of messages from processes that use the same overlay graph or the previous ones, LocalSignaling detects such anomaly and memorizes a negative 'not surviving' result (to be returned at the end of the algorithm). Such process does not stop, but rather keeps signaling using less dense overlay graph, in order to help processes at lower level to survive. This non intuitive behavior is crucial in bounding the amortized bit complexity.

The algorithm proceeds in $O(\log n)$ rounds. Its goal is to leverage the adversary – if the adversary does not fail many processes starting at a level ℓ, some fraction of them will survive and exchange messages in $O(\log n)$ time and $O(\text{polylog } n)$ amortized number of communication bits. To achieve this, a specific family of overlay graphs needs to be used, c.f., Section 4 and Theorem 6.

We will show that if all processes start LocalSignaling at the same time, those who have survived Local Signalling must have had large-size $O(\log n)$-neighborhoods in their communication graph in the beginning of the execution. Moreover, they were able to exchange messages with other surviving processes in their $O(\log n)$-neighborhoods, c.f.. Lemma 17. We will also prove that the amortized bit complexity of the LocalSignaling algorithm is $O(\text{polylog } n)$ per process, c.f., Lemma 16. This is the most advanced technical part used in our algorithm – its full description and analysis are given in Section 8.

3 RELATED WORK

Early work on consensus. The *Consensus* problem was introduced by Pease, Shostak and Lamport [31]. Early work focused on *deterministic* solutions. Fisher, Lynch and Paterson [19] showed that the problem is unsolvable in an asynchronous setting, even if one process may fail. Fisher and Lynch [18] showed that a synchronous solution requires $f + 1$ rounds if up to f processes may crash.

The optimal complexity of consensus with crashes is known with respect to the time and the number of messages (or communication bits) when each among these performance metrics is considered separately. Amdur, Weber and Hadzilacos [4] showed that the amortized number of messages per process is at least constant, even in some failure-free execution. The best deterministic algorithm, given by Chlebus, Kowalski and Strojnowski in [14], solves consensus in asymptotically optimal time $\Theta(n)$ and an amortized number of communication bits per process $O(\text{polylog } n)$.

Efficient randomized solutions against weak adversaries. Randomness proved itself useful to break a linear time barrier for time complexity. However, whenever randomness is considered, different types of an adversary generating failures could be considered.

Chor, Merritt and Shmoys [15] developed constant-time algorithms for consensus against an *oblivious adversary* – i.e., the adversary who knows the algorithm but has to decide which process fails and when before the execution starts. Gilbert and Kowalski [22] presented a randomized consensus algorithm that achieves optimal communication complexity, using $O(1)$ amortized communication bits per process and terminates in $O(\log n)$ time with high probability, tolerating up to $f < n/2$ crash failures.

Randomized solutions against (strong) adaptive adversary. Consensus against an adaptive adversary, considered in this paper, has been already known as more expensive than against weaker adversaries. The time-optimal randomized solution to the consensus problem was given by Bar-Joseph and Ben-Or [9]. Their algorithm works in $O(\frac{\sqrt{n}}{\log n})$ expected time and uses $O(\frac{n^{3/2}}{\log n})$ amortized communications bits per process, in expectation. They also proved optimality of their result with respect to the time complexity, while here we substantially improve the communication.

Beyond synchronous crashes. It was shown that more severe failures or asynchrony could cause a substantially higher complexity. Dolev and Reischuk [16] and Hadzilacos and Halpern [24] proved the $\Omega(f)$ lower bound on the amortized message complexity per process of deterministic consensus for *(authenticated) Byzantine failures*. King and Saia [29] proved that under some limitation on the adversary and requiring termination only whp, the sublinear expected communication complexity $O(n^{1/2}\text{polylog } n)$ per process can be achieved even in case of Byzantine failures. Abraham et al. [1] showed necessity of such limitations to achieve subquadratic time complexity for Byzantine failures.

If *asynchrony* occurs, the recent result of Alistarh et al. [2] showed how to obtain almost optimal communication complexity $O(n \log n)$ per process (amortized) if less then $n/2$ processes may fail, which improved upon the previous result $O(n \log^2 n)$ by Aspnes and Waarts [6]. It is asymptotically almost optimal due to the lower bound $\Omega(n)$ proved by Attiya and Censor-Hillel [7]. Aspnes [5] gave an $\Omega(n/\log^2 n)$ lower bound on the expected number of coin flips.

Fault-tolerant Gossip. was introduced by Chlebus and Kowalski [10]. They developed a deterministic algorithm solving Gossip in time $O(\log^2 f)$ while using $O(\log^2 f)$ amortized messages per process, provided $n - f = \Omega(n)$. They also showed a lower bound $\Omega(\frac{\log n}{\log(n \log n) - \log f})$ on the number of rounds in case $O(\text{polylog } n)$ amortized messages are used per process. In a sequence of papers [10, 11, 21], $O(\text{polylog } n)$ message complexity, amortized per process, was obtained for any $f < n$, while keeping the polylogarithmic time complexity. Note however that general Gossip requires $\Omega(n)$ communication bits per process for different rumors, as each process needs to deliver/receive at least one bit to all non-faulty processes. Randomized gossip against an adaptive adversary is doable w.h.p. in $O(\log^2 n)$ rounds using $O(\log^3 n)$ communication bits per process, for a constant number of rumors of constant size and for $f < \frac{n}{3}$ processes, c.f., Alistarh et al. [3].

4 MODEL AND PRELIMINARIES

In this section we discuss the message-passing model in which all our algorithms are developed and analyzed. It is the classic synchronous message-passing model with processes' crashes, c.f., [8, 9].

Processes. There are n processes with synchronized clocks. Let $\mathcal{P}$ denote the set of all processes. Each process has a unique integer ID in the set $\mathcal{P} = [n] = \{1, \ldots, n\}$. The set $\mathcal{P}$ and its size n are known to all the processes (in the sense that it may be a part of code of an algorithm); it is also called a *KT-1* model in the literature [30].

Communication. The processes communicate among themselves by sending messages. Any pair of processes can directly exchange messages in a round. The point-to-point communication mechanism is assumed to be reliable, in that messages are not lost nor corrupted while in transit.

Computation in rounds. A computation, or an execution of a given algorithm, proceeds in consecutive rounds, synchronized among processes. By a *round* we mean such a number of clock cycles that is sufficient to guarantee the completion of the following operations by a process: first, *multicasting a message* to an arbitrary set of processes (selected by the process during the preceding local computation in previous round or stored in the starting conditions); second, *receiving* the sent messages by their (non-faulty) destination processes; third, performing *local computations*.

Processes' failures and adversaries. Processes may fail by crashing. A process that has *crashed* stops any activity, and in particular does not send nor receive messages. There is an upper bound $f < n$ on the number of crash failures we want to be able to cope with, which is known to all processes in that it can be a part of code of an algorithm. We may visualize crashes as incurred by an omniscient *adversary* that knows the algorithm and has an unbounded computational power; the adversary decides which processes fail and when. The adversary knows the algorithm and is *adaptive* – if it wants to make a decision in a round, it knows the history of computation until that point. However, the adversary does not know the future computation, which means that it does not know future random bits drawn by processes. We do not assume failures to be clean, in the sense that when a process crashes while attempting to multicast a message, then some of the recipients may receive the message and some may not; this aspect is controlled by the adversary. An *adversarial strategy* is a deterministic function, which assigns to each possible history that may occur in any execution some adversarial action in the subsequent round – i.e., which processes to crash in that round and which of their last messages would reach the recipients.

Performance measures. We consider time and bit communication complexities as performance measures of algorithms. For an execution of a given algorithm against an adversarial strategy, we define its time and communication as follows. *Time* is measured by the number of rounds that occur by termination of the last non-faulty process. *Communication* is measured by the total number of bits sent in point-to-point messages by termination of the last non-faulty process. For randomized algorithms, both these complexities are random variables. Time/Communication complexity of a distributed algorithm is defined as a supremum of time/communication taken

over all adversarial strategies, resp. Finally, time/communication complexity of a distributed problem is an infimum of all algorithms' time/communication complexities, resp. In this work we present communication complexity in a form of an *amortized communication complexity* (per process), which is equal to the communication complexity divided by the number of processes n.

Notation whp. We say that a random event occurs *with high probability*, or *whp*, if its probability can be lower bounded by $1 - O(n^{-c})$ for a sufficiently large positive constant c. Observe that when a polynomial number of events occur whp each, then their union occurs with high probability as well.

Overlay graphs. We review the relevant notation and main theorems assuring existence of specific fault-tolerant compact expanders from [14]. We will use them as overlay graphs in the paper, to specify via which links the processors should send messages in order to maintain small time and communication complexities. Some properties of these graphs have already been observed in [14], however we also prove a new property (Lemma 2) and use it for analysis of a novel Local Signalling procedure (Section 8).

Notation. Let $G = (V, E)$ denote an undirected graph. Let $W \subseteq V$ be a set of nodes of G. We say that an edge (v, w) of G is *internal for* W if v and w are both in W. We say that an edge (v, w) of G *connects the sets* W_1 *and* W_2 or *is between* W_1 *and* W_2, for any disjoint subsets W_1 and W_2 of V, if one of its ends is in W_1 and the other in W_2. The *subgraph of* G *induced by* W, denoted $G|_W$, is the subgraph of G containing the nodes in W and all the edges internal for W. A node adjacent to a node v is a *neighbor of* v and the set of all the neighbors of a node v is the *neighborhood of* v. $N_G^i(W)$ denotes the set of all the nodes in V that are of distance at most i from some node in W in graph G. In particular, the (direct) neighborhood of v is denoted $N_G(v) = N_G^1(v)$.

Desired properties of overlay graphs. Let $\alpha, \beta, \delta, \gamma$ and ℓ be positive integers and $0 < \varepsilon < 1$ be a real number. The following definition extends the notion of a lower bound on a node degree:

Dense neighborhood: For a node $v \in V$, a set $S \subseteq N_G^\gamma(v)$ is said to be (γ, δ)-*dense-neighborhood for* v if each node in $S \cap N_G^{\gamma-1}(v)$ has at least δ neighbors in S.

We want our overlay graphs to have the following properties, for suitable parameters α, β, δ and ℓ:

Expansion: graph G is said to be ℓ-*expanding*, or to be an ℓ-*expander*, if any two subsets of ℓ nodes each are connected by an edge.

Edge-density: graph G is said to be (ℓ, α, β)-*edge-dense* if, for any set $X \subseteq V$ of *at least* ℓ nodes, there are at least $\alpha|X|$ edges internal for X, and for any set $Y \subseteq V$ of *at most* ℓ nodes, there are at most $\beta|Y|$ edges internal for Y.

Compactness: graph G is said to be $(\ell, \varepsilon, \delta)$-*compact* if, for any set $B \subseteq V$ of at least ℓ nodes, there is a subset $C \subseteq B$ of at least $\varepsilon\ell$ nodes such that each node's degree in $G|_C$ is at least δ. We call any such set C a *survival set for* B.

Existence of overlay graphs. Let δ, γ, k be integers such that $\delta = 24 \log n, \gamma = 2 \log n$ and $25\delta \leq k \leq \frac{2n}{3}$. Let $G(n, p)$ be an Erdős–Rényi random graph of n nodes, in which each pair of nodes is connected by an edge with probability p, independently over all such pairs.

Theorem 6 ([14]). *For every n and k such that $25\delta \leq k \leq \frac{2n}{3}$, a random graph $G(n, 24\delta/k)$ satisfies all the below properties whp:*
(i) *it is $(k/64)$-expanding,*
(ii) *it is $(k/64, \delta/8, \delta/4)$-edge-dense,*
(iii) *it is $(k, 3/4, \delta)$-compact,*
(iv) *the degree of each node is between $22\frac{n}{k}\delta$ and $26\frac{n}{k}\delta$.*

We define an *overlay graph* $G(n, k, \delta, \gamma)$ as an arbitrary graph of n nodes fulfilling the conditions of Theorem 6. Graph $G(n, k, \delta, \gamma)$ can be computed locally (i.e., in a single round) and deterministically by each process. Specifically, by Theorem 6, the class of graphs satisfying the four properties (i) - (iv) is large, therefore any deterministic search in the class of n-node graphs, applied locally by each process, returns the same overlay graph $G(n, k, \delta, \gamma)$ in all processes.[‡]

Lemma 1 ([14]). *If graph $G = (V, E)$ of n nodes is $(k/64, \delta/8, \delta/4)$-edge-dense then any (γ, δ)-dense-neighborhood for a node $v \in V$ has at least $k/64$ nodes, for $\gamma \geq 2 \lg n$.*

The new property. The key new property of overlay graphs with good expansion, edge-density and compactness is that survival sets in such graphs have small diameters.

Lemma 2. *If graph $G = (V, E)$ of n nodes is $(\frac{k}{64})$-expanding, $(\frac{k}{64}, \frac{\delta}{8}, \frac{\delta}{4})$-edge-dense and $(k, \frac{3}{4}, \delta)$-compact, then for any set $B \subseteq V$ of at least k nodes and for any two nodes v, w from set C being a survival set of B, the nodes v, w are of distance at most $2\gamma + 1$ in graph $G_{|C}$, for any $\gamma \geq 2 \lg n$.*

5 PARAMETERIZED CONSENSUS: TRADING TIME FOR COMMUNICATION

We first specify and analyze algorithm ParameterizedConsensus, for a given parameter $x \in [1, \ldots, n]$[§] and a number of crashes $f < \frac{n}{10}$. Later, in Section 5.4, we show how to generalize it to algorithm ParameterizedConsensus*, which works correctly and efficiently for any number of crashes $f < n$.

Notation and data structures. Let $p \in \mathcal{P}$ denote the process executing the algorithm, while b_p denote p's input bit; $\mathcal{P}, x, p, b_p$ are the input of the algorithm. Let $SP_1, \ldots, SP_x$ be a partition of the set $\mathcal{P}$ of processes into x groups of $\frac{n}{x}$ processes each. SP_i is called a *super-process*, and each $p \in SP_i$ is called its *member*. We also denote by $SP_{[p]}$ the super-process SP_i to whose p belongs. A graph $\mathcal{H}$ is an overlay graph $G(x, \frac{x}{3}, \delta_x, \gamma_x)$, which existence and properties are guaranteed in Theorem 6 and Lemma 2, where $\delta_x := 24 \log x, \gamma_x := 2 \log x$. We uniquely identify vertices of $\mathcal{H}$ with super-processes. We say that two super-processes, SP_p and SP_q, are neighbors if vertices corresponding to them share an edge in $\mathcal{H}$. For every two such neighbors, we denote by $SE(SP_p, SP_q)$ an overlay graph $G(2\frac{n}{x}, \frac{2n}{3x}, 24 \log \frac{2n}{x}, 2 \log \frac{2n}{x})$ which vertices we identify with the set $SP_p \cup SP_q$. ($SE(SP_p, SP_q)$ is a short form of *super-edge* between SP_p and SP_q.) Again, for existence and properties of the above overlay graph we refer to Theorem 6 and Lemma 2. Since the processes operate in KT-1 model, we can assume that all objects mentioned in this paragraph can be computed locally by any process. Alg. 1 gives a pseudo-code of ParameterizedConsensus.

[‡]Recall that each round contributes 1 to the time complexity, no matter of the length of *local* computation.

[§]Without loss of generality, we may assume that x is a divisor of n. If it is not the case, we can always make $\lceil x \rceil$ groups of size $\lceil \frac{n}{x} \rceil$, which would not change the asymptotic analysis of the algorithm.

Algorithm 1: ParameterizedConsensus

input: $\mathcal{P}$, x, p, b_p

1 calculate locally $\{SP_1, \ldots, SP_x\}, \mathcal{H}$;
2 candidate_value ← ParameterizedConsensus:Phase_1($\mathcal{P}$, $\{SP_1, \ldots, SP_x\}$, $\mathcal{H}$, x, p, b_p);
3 confirmed ← ParameterizedConsensus:Phase_2($\mathcal{P}$, $\{SP_1, \ldots, SP_x\}$, $\mathcal{H}$, x, p);
4 **if** confirmed = 1 **then**
5 | CandidatesValues ← Gossip($\mathcal{P}$, p, candidate_value) ; /* Phase 3 */
6 **else**
7 | CandidatesValues ← Gossip($\mathcal{P}$, p,-1); /* Phase 3 */
8 decision_value ← any value in set CandidatesValues that differs from − 1 ;
9 **return** decision_value

High-level idea of ParameterizedConsensus. We cluster processes into x disjoint groups (super-processes) of $\frac{n}{x}$ processes each. Processes locally compute the super-process they belong to and overlay graphs. Starting from this point, we view the system as a set of x super-processes.

In the beginning, Phase 1 is executed (see line 2 of Algorithm 1 and Section 5.1) in which super-processes flood value 1 along an overlay graph $\mathcal{H}$ of super-processes. The main challenge is to do it in $\tilde{O}(\sqrt{xn})$ rounds and $\tilde{O}(\sqrt{n/x})$ amortized communication per process whp.

In Phase 2 (see line 3 and Section 5.2 for description of Phase 2), super-processes estimate the number of operating super-processes in the neighborhood of radius $O(\log x)$ in graph $\mathcal{H}$. Members of those super-processes who estimate at least a certain constant fraction (we say that they "survive"), set up variable confirmed to 1. The main challenge is to do it in $\tilde{O}(\sqrt{n/x})$ rounds and $\tilde{O}(\sqrt{nx})$ amortized communication per process whp.

Next, we discard the partition into x super-processes. All processes execute a Gossip algorithm. Processes that set up variable confirmed to 1 start the Gossip algorithm with their initial value being the value of the super-process they belonged to. Other processes start with a null value (-1). Because super-processes use graph $\mathcal{H}$ for communication, which in particular satisfies $(x, \frac{3}{4}, \delta_x)$-compactness property, we will prove that at the end of Phase 2 at least a constant fraction of non-faulty (i.e., their $\frac{3}{4}$ fraction of members are alive) super-processes survive. This implies that at least a constant fraction of processes begins the Gossip algorithm with a non-null value. Because the non-null value results from a flooding-like procedure of value 1 (if there is any in the system), we will be able to prove that, eventually, every process gets the same value, since at most a constant number of crashes can occur.

To preserve synchronicity, in the ParameterizedConsensus algorithm we use the Monte Carlo version of BiasedConsensus in both Phase 1 and Phase 2, see discussion in Section 2.1. However, with a polynomial small probability, in this variant of Consensus some processes may not reach a decision value. To handle this very unlikely scenario, processes who have not decided in a run of BiasedConsensus alarm the whole system by sending a message to every other process. Then, the whole system switches to any deterministic Consensus algorithm with $O(n)$-time and amortized communication bit complexities (c.f., [8]) and returns its outcome as the final decision. The latter part of alarming and the deterministic Consensus algorithm could use $\Theta(n)$ communication bits, however it happens only with polynomially small probability, see Section 2.1; hence, it does not affect the final amortized complexity of the ParameterizedConsensus algorithm whp. For the sake of clarity, we do not include this straightforward 'alarm' scheme in the pseudocodes.

5.1 Specification and Analysis of Phase 1

Algorithm 2: ParameterizedConsensus:Phase_1

input: $\mathcal{P}$, $\{SP_1, \ldots, SP_x\}$, $\mathcal{H}$, x, p, b_p

1 is_active ← true ;
2 candidate_value ← $\frac{2}{3}$-BiasedConsensus(p, $SP_{[p]}$, b_p);
3 **for** $i \leftarrow 1$ **to** $x + 1$ **do**
4 | **if** is_active = true & candidate_value = 1 **then**
5 | | candidate_value ← $\frac{2}{3}$-BiasedConsensus(p, $SP_{[p]}$, candidate_value);
6 | **else**
7 | | stay silent for $y = \tilde{O}(\sqrt{\frac{n}{x}})$ rounds ;
8 | **if** is_active = true & candidate_value = 1 **then**
9 | | **foreach** *super-process* SP_j *being a neighbor of* $SP_{[p]}$ *in* $\mathcal{H}$ **do**
10 | | | send 1 to every member of SP_j which is a neighbor of p in $SE(SP_{[p]}, SP_j)$;
11 | | **end**
12 | | is_active ← false ;
13 | **if** p *received a message containing* 1 *in the previous round* **then**
14 | | candidate_value ← 1 ;
15 **end**
16 **return** $\frac{1}{3}$-BiasedConsensus(p, $SP_{[p]}$, candidate_value)

High-level idea of Phase 1. In the beginning, the members of every super-process agree on a single value among their input values. Once this is done, super-processes start a flooding procedure navigated by an overlay graph $\mathcal{H}$. $\mathcal{H}$ should be an expander-like, regular graph with good connectivity properties, but a small degree of at most $O(\log x)$. Intuitively, this can guarantee that regardless of the crash pattern there will exist a connected component, of a size being a constant fraction of all vertices, in $\mathcal{H}$ consisting of super-processes that are still operating. The flooding processes is a sequential process of $O(x)$ *phases*. A single super-process *communicates*, that means it sends value 1 to all its neighbors in $\mathcal{H}$, in at most one phase only; either in the first phase, if the value its members agreed on in the beginning is 1; or in the very first phase after the super-process received value 1 from any of its neighbors in $\mathcal{H}$. End of the flooding process encloses the Phase 1 of the algorithm.

Once members of a super-process get value 1 for the first time, they use BiasedConsensus to agree if value 1 has been received or not. It is necessary due to crashes during the flooding procedure,

yet it is not easy to implement with low amortized bit complexity. A pattern of crashes can result in some members of a super-process receiving value 1 and some others not. One can require all members to execute BiasedConsensus in each phase, but this will blow up the amortized bit complexity to $\tilde{O}(x\sqrt{\frac{n}{x}})$ whp. We, in turn, propose to execute BiasedConsensus only among these members who received value 1 in the previous communication round (see line 10) and use the stronger properties of Biased Consensus to argue that the number of calls to the BiasedConsensus will not be to large.

Analysis of Phase 1. Recall, that we say that a super-process *communicates* with another super-process if *any of its members* executes lines 9-11 of the Algorithm 2. Trivially, from the Algorithm 2 we get that each member of a super-process executes line 10 at most once, since if the line is executed then variable `is_active` will be changed to `false`, but the next lemma shows that we can expect more: members of a super-process preserve synchronicity in communicating with other members.

Lemma 3. *For every $i \in [x]$, there is at most one iteration of the main loop in which SP_i communicates with any other super-process.*

Lemma 4. *For every $i \in [x]$, members of a non-faulty super-process SP_i return the same value in Phase_1.*

Recall, that we defined a super-process *non-faulty* if in the end of the `ParameterizedConsensus` algorithm at least $\frac{3}{4}$ of its members have not been crashed. In particular, the number of operating members is at least $\frac{3n}{4x}$ in every phase of the algorithm.

Lemma 5. *There are no two non-faulty super-processes that are connected by an edge in $\mathcal{H}$ but their members return different* `decision_values` *in the end of Phase_1.*

From the previous lemma we can immediately conclude.

Lemma 6. *Members of each connected component of $\mathcal{H}$ formed by a non-faulty super-processes return the same* `decision_values` *in the end of Phase_1.*

Lemma 7. *The Phase_1 part of the ParameterizedConsensus algorithm takes $\tilde{O}(x\sqrt{n/x})$ rounds and uses $\tilde{O}(n\sqrt{n/x}\log n)$ bits whp.*

5.2 Specification and Analysis of Phase 2

High-level idea. In Phase 2, non-faulty super-processes estimate the number of operating super-processes in the neighborhood of radius $O(\log x)$ in graph $\mathcal{H}$. Those who estimate at least a certain constant fraction, set up variable `confirmed` to 1. In order to achieve that, each super-process keeps signaling all its neighbors in $\mathcal{H}$ in $\gamma_x = O(\log x)$ stages until at least a constant fraction of them signaled its activity in preceding stage. A super-process that has been signaling during all stages is said *to survive*. We will prove that, thanks to suitably chosen connectivity properties of $\mathcal{H}$, at least a constant fraction of super-processes survives. Members of these super-processes will influence the final decision of the whole system in the following Phase 3. The following holds:

Lemma 8. *At least $\frac{1}{2}$-fraction of the super-processes are non-faulty and survive Phase_2 of the ParameterizedConsensus algorithm.*

Lemma 9. *The Phase_2 part of the ParameterizedConsensus algorithm takes $\tilde{O}(\sqrt{n/x})$ rounds and uses $\tilde{O}(n\sqrt{n/x})$ bits whp.*

Algorithm 3: ParameterizedConsensus:Phase_2

```
   input: P, {SP_1, ..., SP_x}, H, x, p
1  if 3/4-BiasedConsensus(p, SP_[p], 1) = 1 then
2  |  is_active ← true
3  else
4  |  is_active ← false ;                    /* stage i */
5  for i ← 1 to γ_x do
6  |  if is_active = true then
7  |  |  SN ← ∅ ;
8  |  |  foreach super-process SP_j being a neighbor of SP_[p]
   |  |   in H do
9  |  |  |  N_j ← Gossip(SP_[p] ∪ SP_j, p, p) ;
10 |  |  |  SN ← SN ∪ N_j ;
11 |  |  end
12 |  |  if |SN| > δ_x then many_superprocesses ← 1 ;
13 |  |  else many_superprocesses ← 0 ;
14 |  |  survived ← 2/3-BiasedConsen-
   |  |   sus(p, SP_[p], many_superprocesses) ;
15 |  |  if survived = 0 then
16 |  |  |  is_active ← false
17 |  |  end
18 end
19 return is_active; /* p's super-process survived? */
```

5.3 Analysis of ParameterizedConsensus

Lemma 10. *The value* `candidate_value` *is the same among all members of super-processes that survived Phase_2.*

Lemma 11. *The algorithm ParameterizedConsensus satisfies validity, agreement and termination conditions.*

Theorem 7. *For any $x \in [1, n]$ and any number of crashes $f < \frac{n}{10}$, ParameterizedConsensus solves Consensus with probability 1, in $O(\sqrt{nx}\ \text{polylog}\, n)$ time and $O(\sqrt{n/x}\ \text{polylog}\, n)$ amortized bit communication complexity, whp, using $O(\sqrt{n/x}\ \text{polylog}\, n)$ random bits per process.*

Proof. By Lemma 11 we already know that the ParameterizedConsensus algorithm is a solution to the Consensus problem.

By Lemma 7 and Lemma 9 we get the time and bit complexity of Phase_1 and Phase_1. By Theorem 3, we have that a single execution of a Gossip algorithm takes $\tilde{O}(1)$ rounds and $\tilde{O}(1)$ communication bits amortized per process, given that there can be only two different rumors of size $\tilde{O}(1)$ each, as we argued in Lemma 11. These bounds together give us the desired complexity of the ParameterizedConsensus algorithm.

A single run of the α-BiasedConsensus algorithm on members of a super-processes generates $\tilde{O}(\frac{n}{x}\sqrt{\frac{n}{x}})$ random bits, since each member generates at most one random bit per every round of the algorithm, see Section 2.1. Since, the processes execute at most $\tilde{O}(x)$ runs of the α-BiasedConsensus algorithm, thus the total number of random bits used is $\tilde{O}(n\sqrt{\frac{n}{x}})$ which implies $\tilde{O}(\sqrt{\frac{n}{x}})$ amortized random bit complexity. □

5.4 Generalization to Any Number of Failures.

In this subsection we highlight main ideas that generalize the ParameterizedConsensus algorithm to work in the presence of *any* number of crashes $f < n$. We call the resulting algorithm ParameterizedConsensus*. We exploit the concept of epochs in a similar way to [9, 14]. In short, the first and main epoch (in our case, ParameterizedConsensus followed by BiasedConsensus described in Section 2.1) is repeated $O(\log n)$ times, each time adjusting expansion/density/probability parameters by factor equal to $\frac{9}{10}$. The complexities of the resulting algorithm are multiplied by logarithmic factor. More details are given below.

Consider a run of the ParameterizedConsensus algorithm, as described and analyzed in previous sub-sections. Let us analyze the state of the system at the end of ParameterizedConsensus algorithm if more than $\frac{n}{10}$ crashes have occurred. In the end, there exist two group of processes, those that have `decision_value` set to -1 (i.e., the last Gossip has not been successful in their case), and those who have `decision_value` set to a value from $\{0, 1\}$. Observe, that if at most $\frac{n}{10}$ processes were faulty, then we already proved in Theorem 7 that the first of these sets would be empty and there could be only one value in $\{0, 1\}$ taken by alive processes. Thus, we can extend the run of the ParameterizedConsensus by an execution of $\frac{1}{2}$-BiasedConsensus among members of each super-processes, separately for different super-processes, to make them agree if there exists a member of the super-process who had received a null value in the last Gossip execution. A single run of ParameterizedConsensus followed by the run of $\frac{1}{2}$-BiasedConsensus is called an *epoch*. Based on the output of the $\frac{1}{2}$-BiasedConsensus, the members of each super-process decide whether they keep the agreed candidate value as decision final value and stay idle in the next epoch, or they continue to the next epoch. There are three key properties here. First, because the decision of entering next epoch is made based on an output to Biased Consensus, it is consistent among members of a single super-process. Second, in the good scenario, i.e., when only less than $\frac{n}{10}$ processes crashed, every process will start the run of the $\frac{1}{2}$-BiasedConsensus with the same value, yet different than a null-value. From validity condition, all processes stay idle. Third, a non-faulty super-process at the end of Phase 2 actually implies that there was a majority of non-faulty other super-processes in its $O(\log n)$ neighborhood, regardless of the number of failures (c.f., Lemma 17 – thus, only one value in $\{0, 1\}$ can be confirmed in the whole system as long as at least one process remains alive, whp.

In the next epoch, super-processes that are not idle, repeat the ParameterizedConsensus algorithm, but tune its parameters to adjust to the larger number of crashes (i.e., smaller fraction of alive processes). They use:

(i) a graph $\mathcal{H}_1$, instead of $\mathcal{H}$, which is roughly $\frac{10}{9}$ denser (i.e., a graph $G(\cdot)$) compared to graph $\mathcal{H}$ used in the previous epoch,
(ii) new threshold $\alpha_1 := \frac{2}{3} \cdot \frac{9}{10}$ for evoking BiasedConsensus,
(iii) they loose the parameter in the definition of a non-faulty super-process by a factor of $9/10$.

In general, processes repeats this process of 'densification' in subsequent $\Theta(\log n)$ epochs. Eventually, one of this epochs must be successful, otherwise the number of crashed process would exceed $n/(1/10)^{\Theta(n)} > n$. On the other hand, each time we 'densify' graph $\mathcal{H}$, i.e., we take an overlay graph $\mathcal{H}_i$ from the family of overlay graphs as defined in Section 4 but with expansion and density parameters adjusted by factor $\left(\frac{9}{10}\right)^i$, we are guaranteed that only a fraction of previously alive processes execute the next epoch. As density and expansion parameters in the family of overlay graphs are inversely proportional, we conclude that in each epoch the amortized bit complexity stays at the same level of $O(\sqrt{\frac{n}{x}})$. Therefore, in cost of multiplying both, the time complexity and the amortized bit complexity by a factor of $\Theta(\log n)$, we are able to claim Theorem 1.

Theorem 1 (Strengthened Theorem 7). *For any $x \in [1, n]$ and the number of crashes $f < n$, ParameterizedConsensus* solves Consensus with probability 1, in $O(\sqrt{nx}\ polylog\ n)$ time and $O(\sqrt{n/x} \cdot polylog\ n)$ amortized bit communication complexity, whp, while using $O(\sqrt{n/x}\ polylog\ n)$ random bits per process.*

6 RANDOMIZED α-BIASED CONSENSUS

The α-BiasedConsensus algorithm generalizes and improves the SynRan algorithm of Bar-Joseph and Ben-Or [9]. For this part, we purposely use the same notation as in [9] for the ease of comparison.

First, processes run Fuzzy Counting (i.e. use the FuzzyCounting algorithm from Section 7) where the set of active processes consists of this processes which the input value to the α-Biased Consensus is 1. Then, each process calculates logical *AND* of the two values: its initial value and $\texttt{ones} \geq \alpha \cdot n$, where ones is the number of 1's output by the Fuzzy Counting algorithm. Denote x_p the output of the logical *AND* calculated by process p.

In the following processes solves an α-Biased Consensus on x_p. Each process p starts by setting its current choice b_p to x_p. The value b_p in the end of the algorithm indicates p's decision. Now, processes use $O(f/\sqrt{n \log n})$ *phases* to update their values b_p such that eventually every process keeps the same choice. To do so, in a round r every process p calculates the number of processes that current choice is 1 and the number of processes that current choice is 0, denoted O_p^r and Z_p^r respectively. Based on these numbers, process p either sets b_p to 1, if the number O_p^r is large enough; or it sets b_p to 0, if the number Z_p^r is large; or it replaces b_p with a random bit, if the number of zeros and ones are close to each other. In Bar-Joseph's and Ben-Or's the numbers Z_p^r and O_p^r were calculate in a single round all-to-all of communication. However, we observed that because processes' crashes may affect this calculation process in almost arbitrary way, this step can be replaced by any solution to Fuzzy Counting. That holds, because Fuzzy Counting exactly captures the necessary conditions that processes must fulfill to simulate the all-to-all communication, that is it guarantees that candidate values of non-faulty processes are included in the numbers O_p^r and Z_p^r calculated by any processor p. Thus, rather than using all-to-all communication, our algorithms utilizes the effective FuzzyCounting algorithm where active processes are those who have their current choice equal 1. The output of this algorithm serves as the number O_p^r, while the number Z_p^r is just $n - O_p^r$. For the sake of completeness, we also provide the pseudocode of the algorithm. We conclude the above algorithm in Theorem 2.

Algorithm 4: α-BiasedConsensus. The parts in which our algorithm differs from the SynRyn algorithm from [9] algorithm are underlined.

input: $\mathcal{P}$, p, b_p, α
if FuzzyCounting$(\mathcal{P}, p, b_p) > \alpha \cdot |\mathcal{P}|$ **then** $x_p \leftarrow b_p$ & 1;
else $x_p \leftarrow 0$;
$r \leftarrow 1$; $N^r_{-1} = N^r_0 \leftarrow n$; decided $\leftarrow$ *FALSE* ;
while *TRUE* **do**
  participate in CheapCounting execution with input bit being set to b_p; let O^r_p, Z^r_p be the numbers of ones and zeros (resp.) returned by CheapCounting;
  $N^r_p \leftarrow Z^r_p + O^r_p$;
  if $(N^r_p < \sqrt{n/\log n})$ **then**
    send b_p to all processes, receive all messages sent to p in round $r+1$;
    run any deterministic Consensus protocol on the set $\mathcal{P}$ of all processes, working in at most $\sqrt{n/\log n}$ rounds and using all-to-all communication, c.f., [8];
  end
  if decided = *TRUE* **then**
    diff $\leftarrow N^{r-3}_p \, N^r_i$;
    if (diff $\leq N^{r-2}_p/10$) **then** STOP;
    else decided $\leftarrow$ *FALSE*;
  end
  if $O^r_p > (7N^r_p - 1)/10$ **then** $b_p \leftarrow 1$, decided $\leftarrow$ *TRUE*;
  else if $O^r_p > (6N^r_p - 1)/10$ **then** $b_p \leftarrow 1$;
  else if $Z^r_p = 0$ **then** $b_p \leftarrow 1$;
  else if $O^r_p < (4N^r_p - 1)/10$ **then**
    $b_p \leftarrow 0$, decided $\leftarrow$ *TRUE*;
  else if $O^r_p < (5N^r_p - 1)/10$ **then** $b_p \leftarrow 0$;
  else set b_p to 0 or 1 with equal probability;
  $r \leftarrow r + 1$;
end
return b_p; /* consensus value */

Theorem 2. *The α-BiasedConsensus algorithm solves α-Biased Consensus with probability 1. The algorithm has expected running time $O(f/\sqrt{n} \cdot \log^{5/2} n)$ and expected amortized bit complexity $O(f/\sqrt{n} \cdot \log^{13/2} n)$, for any number of crashes $f < n$.*

Setting $\alpha := \frac{1}{2}$ we get a better randomized solution to classic Consensus problem.

Corollary 1. *The $\frac{1}{2}$-BiasedConsensus algorithm is a solution to Consensus. The algorithm satisfies agreement and validity with probability 1, has expected running time $O(f/\sqrt{n} \cdot \log^{5/2} n)$, and the expected amortized bit complexity $O(f/\sqrt{n} \cdot \log^{13/2} n)$, for any number of crashes $f < n$.*

Monte Carlo version. The original algorithm α-BiasedConsensus has the expected running time $O(\sqrt{n} \log^{13/2} n)$. However, we can force all processes to stop by that time multiplied by a constant. In such case, the worst-case running time will be always $\tilde{O}(\sqrt{n})$ while the correctness (agreement) will hold only whp.

7 GOSSIP AND FUZZY COUNTING

In this section we design and analyze an algorithm, called Gossip which, given a set of processes $\mathcal{P}$, solves the Gossip problem in $\tilde{O}(1)$ rounds and uses $\tilde{O}(|\mathcal{R}|)$ communication bits amortized per process, where $|\mathcal{R}|$ is the number of bits needed to encode initial rumors of all processes. A small modification of this algorithm will result in a solution to the Fuzzy Counting problem with the same time and only logarithmically larger bit complexity.

7.1 Bipartite Gossip

We start by giving a solution to Gossip problem in a special case, called *Bipartite Gossip*, in which processes are partitioned into two groups $\mathcal{P}_1$ and $\mathcal{P}_2$ each of size $\lceil n/2 \rceil$ at most. Processes starts with at most two different initial rumors r_1 and r_2 such that processes of each group share the same initial rumor. The partition and the initial rumor is assumed to be an input to the algorithm. The goal of the system is still to achieve Gossip.

High level idea of algorithm BipartiteGossip. If there were no crashes in the system, it would be enough if processes span a bipartite expanding graph with poly-logarithmic degree on the set of vertices $\mathcal{P}_1 \cup \mathcal{P}_2$ and for $\tilde{O}(1)$ rounds exchange messages with their initial rumors. In this ideal scenario the $O(\log n)$ bound on the expander diameter suffices to allow every two process exchange information, while the sparse nature of the expander graphs contributes to the small bit complexity. However, a malicious crash pattern can easily disturb such naive approach. To overcome this, in our algorithm processes will adapt to the number of crashes they estimate in their group, by communicating over denser expander graphs from a family of $\Theta(\log n)$ expanders, every time they observe a significant reduction of non-faulty processes in their neighborhood.

Precisely, the *internal* communication within group P_1 uses graphs from a family of $\Theta(\log n)$ expanders:
$\mathcal{G}_{\mathtt{in}} = \{G_{\mathtt{in}}(0), \ldots, G_{\mathtt{in}}(\log n)\}$, for $t = O(\log n)$, spanned on the set of processes $\mathcal{P}_1$ and such that $G_{\mathtt{in}}(i) \subseteq G_{\mathtt{in}}(i+1)$, the degree and expansion parameter of the graphs double with the growing index, and the last graph is a clique. Initially, processes from $\mathcal{P}_1$ span an expander graph $G_{\mathtt{in}}(0)$ with $O(\log n)$ degree on the set $\mathcal{P}_1$, in the sense that each process in $\mathcal{P}_1$ identifies its neighbors in the graph spanned on $\mathcal{P}_1$. In the course of an execution, each process from $\mathcal{P}_1$ keeps testing the number of non-faulty processes in its $O(\log n)$ neighborhood in $G_{\mathtt{in}}(0)$. If the number falls down below some threshold, the process upgrades the used expanding graph by switching to the next graph from the family – $G_{\mathtt{in}}(1)$. The process continues testing, and switching graph to the next in the family if necessary, until the end of the algorithm. The ultimate goal of this 'densification' of the overlay graph is to enable each process' communication with a constant fraction of other alive processes in $\mathcal{P}_1$. Note here that this procedure of adaptive adjustment to failures pattern happens independently at processes in $\mathcal{P}_1$, therefore it may happen that processes in $\mathcal{P}_1$ may have neighborhoods taken from different graphs in family $\mathcal{G}_{\mathtt{in}}$.

The *external* communication of processes from $\mathcal{P}_1$ with processes from $\mathcal{P}_2$ is strictly correlated with their estimation of the number of processes being alive in their $O(\log n)$ neighborhood in $\mathcal{P}_1$ using expanders in $\mathcal{G}_{\mathtt{in}}$, as described above. Initially, a process from $\mathcal{P}_1$

sends its rumor according to other expander graph $G_{\mathtt{out}}(0)$ of degree $O(\log n)$, the first graph in another family of expanders graphs $\mathcal{G}_{\mathtt{out}} = \{G_{\mathtt{out}}(0), \ldots, G_{\mathtt{out}}(t)\}$, for $t = O(\log n)$, spanned on the whole set of processes $\mathcal{P}_1 \cup \mathcal{P}_2$, such that $G_{\mathtt{out}}(i) \subseteq G_{\mathtt{out}}(i+1)$, the degree and expansion parameter of the graphs double with the growing index, and the last graph is a clique. Each time a process chooses a denser graph from family $\mathcal{G}_{\mathtt{in}}$ in the internal group communication, described in the previous two paragraphs, it also switches to a denser graph from family $\mathcal{G}_{\mathtt{out}}$ in the external communication with group $\mathcal{P}_2$. The intuition is that if a process knows that the number of alive processes in its $O(\log n)$ neighborhood in $\mathcal{P}_1$ has been reduced by a constant factor since the last check, it can afford an increase of its degree in external communication with group $\mathcal{P}_2$ by the same constant factor, as the amortized message complexity should stay the same.

Estimating the number of alive processes in $O(\log n)$ neighborhoods. In the heart of the above method lies an algorithm, called LocalSignaling that for each process p, tests the number of other alive processes in p's neighborhood of radius $O(\log n)$. As a side result, it also allows to exchange a message with these neighbors. The algorithm takes as in input: a set of all processes in the system $\mathcal{P}$, an expander-like graph family $\mathcal{G} = \{G(0), \ldots, G_t\}$ spanned on $\mathcal{P}$, together with two parameters δ and γ, describing a diameter and a maximal degree of the base graph $G(0)$; the name of a process p; the process' level ℓ which denotes which graph from family $\mathcal{G}$ the process uses to communicate; and the message to convey r. Let $\mathcal{T}$ denote a graph $\cup_{v \in \mathcal{P}} N_{G_{\ell_v}}(v)$, that is a graph with set of vertices corresponding to $\mathcal{P}$ and set of edges determined based on neighbors of each vertex from a graph on the proper level. Provided that LocalSignaling is executed synchronously on the whole system it returns whether the process p was connected to a constant number of other alived processes at the beginning of the execution accordingly to graph $\mathcal{T}$. Assumed that, the algorithm guarantees that p's message reached all these processes and vice versa - messages of these processes reached p. On the other hand, we will prove that the amortized bit complexity of a synchronous run of the LocalSignaling algorithm is $\tilde{O}(n)$. This is the most advanced technical part used in our algorithm. It's full description and detailed analysis is given in Section 8.

BipartiteGossip *algorithm and its analysis.* In this paragraph we give a pseudocode of the BipartiteGossip algorithm which implements the idea discussed before. We start by formal description of utilized graphs and connected to them subroutines.

The graphs used by processes are grouped into two families: $\mathcal{G}_{\mathtt{in}}$ and $\mathcal{G}_{\mathtt{out}}$. Denote $t = \lfloor \log n \rfloor$, $\delta = 2 \log n$, $\gamma = 24 \log n$. Consider a process p; it gets as an input the partition of set $[n]$ into groups P_1, P_2, thus it can determine the group it belongs to. The family $\mathcal{G}_{\mathtt{in}} = \{G_{\mathtt{in}}(0), \ldots, G_{\mathtt{in}}(t+1)\}$ serves for communication *inside* each group.

A single graph $G_{\mathtt{in}}(i)$, for $i \in \{0, \ldots, t\}$, is a union of $G(n/2, \frac{n}{3 \cdot 2^j}, \delta, \gamma)$, over $j \in \{0, \ldots, i\}$, of graphs given in the Theorem 6 with nodes being the processes in p's group, that is $G_{\mathtt{in}}(i) = \bigcup_{j=0}^{j=i} G(n/2, \frac{n}{3 \cdot 2^j}, \delta, \gamma)$. Graph G_{t+1} is a clique with nodes being the processes of p's group.

The family $\mathcal{G}_{\mathtt{out}} = \{G_{\mathtt{out}}(0), \ldots, G_{\mathtt{out}}(t+1)\}$ serves for communication *outside* each group. A single graph $G_{\mathtt{out}}(i)$, for $i \in \{0, \ldots, t\}$, is a union of $G(n, \frac{2n}{3 \cdot 2^j}, \delta, \gamma)$, over $j \in \{0, \ldots, i\}$, of graphs given in the Theorem 6 with nodes being all the processes, that is $G_{\mathtt{out}}(i) = \bigcup_{j=0}^{j=i} G(n, \frac{2n}{3 \cdot 2^j}, \delta, \gamma)$. Graph G_{t+1} is a clique with nodes being all the processes.

Observe, that those families and parameters t, δ, γ are deterministic and can be precomputed by each process, assumed the knowledge of partition P_1 and P_2. As a such, they are assumed to be known to the algorithm on every stage of the algorithm.

The Exchange communication scheme for a graph G, used in the BipartiteGossip algorithm: This communication scheme takes two rounds. In the first round p sends a message containing a bit and the set R, being a set of all learned so far rumors by p, to every process in the set $N_G(p)$ that is not faulty according to p's view on the system. The receiver treats such a message as both a request and an increment-knowledge message. In the second round of the Exchange, p responds to all the received requests by sending R to each sender of every request received in the previous round.

Algorithm 5: BipartiteGossip

```
   input: partition P_1, P_2; p, r, R = {r}
1  for i ← 1 to 2t do
2      repeat 3 times
3          do Exchange on graph G_out(i + 1);
4          repeat 2γ + 1 times
5              do Exchange on graph G_in(i + 7);
6          repeat t + 2 times
7              do Exchange on graph G_in(i + 2);
8              survived ← LocalSignaling(p, G_in, i, δ, γ, R);
9              if survived = false then
10                 i ← min(i + 1, t + 1)
11             end
12 end
13 return R;                /* set R of learned rumors */
```

Analysis of correctness. We call a single iteration of the main loop of the BipartiteGossip algorithm an *epoch*. First, we show that if in a single epoch a big fraction of processes from the groups P_1 and P_2 worked correctly, then by the end of the epoch every process has learned both rumors r_1 and r_2. Let $\mathcal{E}$ be an epoch. Let $\mathtt{BEGIN}_1$ ($\mathtt{BEGIN}_2$) be the set of processes from the group P_1 (group P_2 respectively) that were non-faulty before the epoch $\mathcal{E}$ started. Let $\mathtt{END}_1$ ($\mathtt{END}_2$) be the set of those processes from the group P_1 (group P_2 respectively) that were non-faulty after the epoch $\mathcal{E}$ ended. We assume that epoch $\mathcal{E}$ is such that:

$$|\mathtt{END}_1| > \tfrac{1}{3}|\mathtt{BEGIN}_1| \quad \text{and} \quad |\mathtt{END}_2| > \tfrac{1}{3}|\mathtt{BEGIN}_2| \,.$$

Lemma 12. *After the first iteration of the loop from line 2 in epoch $\mathcal{E}$, each non-faulty process from the group P_1 is on level $j_p \geq \log\left(\frac{n}{3 \cdot 64 \cdot |\mathtt{BEGIN}_1|}\right)$.*

Lemma 13. *There exists a set $C_1 \subseteq \mathtt{END}_1$ of size at least $\frac{|\mathtt{BEGIN}_1|}{4}$ such that after the second iteration of the loop 2 of epoch $\mathcal{E}$ each process p from set C_1 has the other rumor r_2 in its set $\mathcal{R}$.*

Lemma 14. *After the epoch $\mathcal{E}$ ends, each process from the set $\mathtt{END}_1$ knows the other rumor r_2.*

Analysis of communication complexity. Let $L_i(r)$ be the set of non-faulty processes that at the beginning of the round r are on level i or bigger. We show that for any round $r \geq 2$ and for any $i \in [t]$, the number $|L_i(r)|$ is at most $\frac{2n}{2^i}$.

Lemma 15. *For any round $r \geq 2$ and any level $i \in [t]$ the number of processes in the set $L_i(r)$ is at most $\frac{2n}{2^i}$.*

Putting the above Lemmas together, Theorem 5 could be proved.

7.2 The Gossip Algorithm

Here, we describe an algorithm based on the divide-and-conquer approach, called Gossip that utilizes the BipartiteGossip algorithm to solve Fault-tolerant Gossip. Each process takes the set $\mathcal{P}$, an initial rumor r and its unique name $p \in [|\mathcal{P}|]$ as an input. The processes split themselves into two groups of size at most $\lceil n/2 \rceil$. The groups are determined based on the unique names. The first $\lceil n/2 \rceil$ processes with the smallest names make the group $\mathcal{P}_1$, while the $n - \lceil n/2 \rceil$ processes with the largest names define the group $\mathcal{P}_2$. Each of those two groups of processes solves Gossip separately by evoking the Gossip algorithm inside the group only. The processes from each group know the names of every other process in that group, hence the necessary conditions to execute the Gossip recursively are satisfied. After the recursion finishes, a process from $\mathcal{P}_1$ stores a set of rumors $\mathcal{R}_1$ of processes from its group, and respectively a process from $\mathcal{P}_2$ stores a set of rumors $\mathcal{R}_2$ of processes from its group. Then, the processes solve Bipartite Gossip problem by executing the BipartiteGossip algorithm on the partition $\mathcal{P}_1, \mathcal{P}_2$ and having initial rumors $\mathcal{R}_1$ and $\mathcal{R}_2$. The output to this algorithm is the final output of the Gossip, for which Theorem 3 holds.

Modification for Fuzzy Counting. We define the *Fuzzy Counting* problem as follows. There is a set n processes, $\mathcal{P}$, with unique names that are comparable. Each process knows the names of other processes (i.e. they operate in KT-1 model). Each process starts with an initial bit $b \in \{0, 1\}$. Let $\mathtt{Z}$ denote the number of processes that started with the initial bit set to 0 and never failed. Similarly, $\mathtt{O}$ denotes the number of processes that started with 1 and never failed. Each process has to return two numbers: `zeros` and `ones`. An algorithm is said to solve fuzzy counting if every non faulty process terminates (termination condition) and the values returned by any process fulfill the conditions: $\mathtt{zeros} \geq |\mathtt{Z}|$, $\mathtt{ones} \geq |\mathtt{O}|$ and $\mathtt{zeros} + \mathtt{ones} \leq n$ (validity condition).

To solve this problem, we use the Gossip algorithm with the only modification that now we require the algorithm the return the values $\mathtt{Z}$ and $\mathtt{O}$, instead of the set of learned rumors. We apply the same divide-and-conquer approach. That is, we partition $\mathcal{P}$ into groups $\mathcal{P}_1$ and $\mathcal{P}_2$ and we solve the problem within processors of this partition. Let $\mathtt{Z}_1$, $\mathtt{O}_1$ and $\mathtt{Z}_2$, $\mathtt{O}_2$ be the values returned by recursive calls on set of processes $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Then, we use the BipartiteGossip algorithm to make each process learn values $\mathtt{Z}$ and $\mathtt{O}$ of the other group. Eventually, a process returns a pair of values $\mathtt{Z}_1 + \mathtt{Z}_2$ and $\mathtt{O}_1 + \mathtt{O}_2$ if it received the values from the other partition during the execution of BipartiteGossip; or it returns the values corresponding to the recursive call in its partition otherwise. It is easy to observe, that during this modified execution processes must carry messages that are able to encode values $\mathtt{Z}$ and $\mathtt{O}$, thus in this have it holds that $|\mathcal{R}| = O(\log n)$. The above modification leads to Theorem 4.

8 LOCAL SIGNALLING – ESTIMATING NEIGHBORHOODS IN EXPANDERS

The LocalSignaling algorithm, presented in this section, allows to adapt the density of used overlay graph to any malicious fail pattern guaranteeing fast information exchange among a constant fraction of non-faulty nodes with amortized $\tilde{O}(n|\mathcal{R}|)$ bit complexity, where $\mathcal{R}$ is the overhead that comes from the bit size of the information needed to convey.

It is formally denoted $\mathtt{LocalSignaling}(\mathcal{P}, p, \mathcal{G}, \delta, \gamma, \ell, r)$, where $\mathcal{P}$ is the set of all processes, p is the process that executes the procedure and $\mathcal{G} = \{G(1), \ldots, G(t)\}$ denotes the family of overlay graphs that processes from $\mathcal{P}$ uses to select processes to directly communicate – those are neighborhoods in some graph of the family $\mathcal{G}$. In our case, the family will consist of graphs with increasing connectivity properties. Parameters γ, δ correspond to the property of (γ, δ)-dense-neighborhoods which the base graph $G(1)$ must fulfill. They are also related to the time and actions taken by processes if failures occur, respectively. The parameter $\ell \leq t$ is called a *starting level* of process p and denotes the communication graph from family $\mathcal{G}$ from which the node p starts the current run of the procedure. This parameter may be different for different processes. Finally, the parameter r denotes a rumor that process p is supposed to deliver to other processes. Since processes operates in KT−1 model, the implementation assumes that each process uses the same family $\mathcal{G}$ (see the corresponding discussion after Theorem 6).

LocalSignaling$(\mathcal{P}, p, \mathcal{G}, \delta, \gamma, \ell, r)$ takes 2γ consecutive rounds. The level of process p executing the procedure is initially set to ℓ, and is stored in a local variable i. Each process stores also a set R of all rumors it has learned to this point. Initially, R is set to $\{r\}$.

Odd rounds: Process p sends a request message to each process q in $N_{G(i)}(p)$, provided $i > 0$.

Even rounds: Every non-faulty process q responds to the requests received at the end of the previous round – by replying to the originator of each request a message containing the current level i of process q and the set R of all different rumors q collected so far. At the end of each even round, processes that requested information in the previous round collect the responses to those requests. If a single process p received less then δ responses with level's value of its neighbors greater or equal than its level value i, then p decreases i by one. Additionally, p merges every set of rumors it received with its own set R. If i drops to 0, then p does not send any requests in the consecutive rounds.

Output: We say that process p *has not survived* the LocalSignaling algorithm if it ends with value i lower than its initial level i. Otherwise, p is said to have *survived* the LocalSignaling algorithm. p returns a single bit indicating whether it has survived or not and the set R containing all rumors it has learnt in the course of the execution.

Lemma 16. *The procedure LocalSignaling$(\mathcal{P}, p, \mathcal{G}, \delta, \gamma, \ell, r)$ takes $O(\gamma)$ rounds and uses $\left(\sum_{i=1}^{i=t} |L_i| \cdot |N_{G_{\leq i}}(L_i)| \cdot \gamma \cdot |\mathcal{R}|\right)$ communication bits, where L_i denotes the set of processes that start at level i, the graph*

$G_{\leq i}$ is a union of graphs $G(1), \ldots, G(i)$, and the value $|\mathcal{R}|$ denotes the number of bits needed to encode all possible rumors.

Surviving the LocalSignaling – the consequences. Assume that $t \geq 1$ and consider a sequence $(k_i)_{i\in[t]}$. Let $\mathcal{G} = \{G(1), \ldots, G(t)\}$ be a family of graphs $G(i) = G(n, k_i, \delta, \gamma)$ defined as in Theorem 6. We require, for any $1 \leq i < t$ that $G(i) \subseteq G(i+1)$. Consider a simultaneous run of the procedure LocalSignaling$(\mathcal{P}, p, \mathcal{G}, \delta, \gamma, \ell, r)$ at every process $p \in \mathcal{P}$. Here, we require each process $p \in \mathcal{P}$ to use the same family of graphs $\mathcal{G}$. Since our processes operates in KT–1 model, this requirement could be always satisfied.

Let $B_{\ell,1}$ be *the start set on level ℓ*: it consists of the processes that are non-faulty at the beginning of this instance of Local Signaling and their level is at least ℓ. Let $B_{\ell,2} \subseteq B_{\ell,1}$ be *the end set*: it consists of the processes that are non-faulty just after the termination of this instance and their level at the beginning of this instance was at least ℓ. The processes in $B_{\ell,1} \setminus B_{\ell,2}$ are among those that have crashed during the considered instance of Local Signaling.

Lemma 17. *The following properties hold for arbitrary times of crashes of the processes in $B_{\ell,1} \setminus B_{\ell,2}$:*

1. If there is a (γ, δ)-dense-neighborhood for $p \in B_{\ell,2}$ in graph $G_\ell|_{B_{\ell,2}}$, then process p survives Local Signaling.

2. If p survived the Local Signaling, then there is (γ, δ)-dense-neighborhood for $p \in B_{\ell,1}$ in graph $G(\ell)|_{B_{\ell,1}}$. Moreover, p receives the rumor r of any node from that (γ, δ)-dense-neighborhood.

3. Any process in a survival set C for $B_{\ell,2}$ that started at level exactly ℓ survives Local Signaling.

9 CONCLUSIONS AND OPEN PROBLEMS

We explored the Consensus problem in the classic message-passing model with processes' crashes, from perspective of both time *and* communication optimality. We discovered an interesting tradeoff between these two complexity measures: Time × Amortized_Communication = $\tilde{O}(n)$, which, to the best of our knowledge, has not been present in other settings of Consensus and related problems. We believe that a corresponding lower bound could be proved: Time × Amortized_Communication = $\tilde{\Omega}(n)$. Interestingly, a similar tradeoff could hold between time and amount of randomness, as our main algorithm ParameterizedConsensus* satisfies the relation: Time × Amortized_Randomness = $\tilde{O}(n)$. Exploring similar tradeoffs in other fault-tolerant distributed computing problems could be a promising and challenging direction to follow. It is worth noting that all our algorithms use messages of size $O(\log n)$, and thus can be implemented in the congest model.

ACKNOWLEDGMENTS

M.T. HajiAghayi and J. Olkowski were partially supported by NSF CCF grant-2114269 and an Amazon AWS award. D.R. Kowalski was partially supported by the NSF grant 2131538.

REFERENCES

[1] I. Abraham, T.-H. Hubert Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, Communication Complexity of Byzantine Agreement, Revisited, in *Proc., ACM Symposium on Principles of Distributed Computing (PODC)*, 2019, pp. 317 - 326.
[2] D. Alistarh, J. Aspnes, V. King, and J. Saia, Communication-efficient randomized consensus, *Distributed Comput.*, 31 (2018), 489 - 501.
[3] D. Alistarh, S. Gilbert, R. Guerraoui, M. Zadimoghaddam, How Efficient Can Gossip Be? (On the Cost of Resilient Information Exchange), *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010*, 115 - 126.
[4] S. Amdur, S. Weber, and V. Hadzilacos, On the message complexity of binary agreement under crash failures, *Distributed Computing*, 5 (1992) 175 - 186.
[5] J. Aspnes, Lower Bounds for Distributed Coin-Flipping and Randomized Consensus, *J. ACM* 45(3) (1998): 415 - 450.
[6] J. Aspnes and O. Waarts, Randomized Consensus in Expected $O(nlog^2n)$ Operations Per Processor, *SIAM J. Computing* 25(5) (1996): 1024 - 1044.
[7] H. Attiya and K. Censor-Hillel, Lower Bounds for Randomized Consensus under a Weak Adversary, *SIAM J. Comput.* 39(8) (2010): 3885–3904.
[8] H. Attiya, and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, 2nd edition, Wiley, 2004.
[9] Z. Bar-Joseph, and M. Ben-Or, A Tight Lower Bound for Randomized Synchronous Consensus, in *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1998, pp. 193 - 199.
[10] B.S. Chlebus, and D.R. Kowalski, Robust gossiping with an application to consensus, *Journal of Computer and System Sciences*, 72 (2006) 1262 - 1281.
[11] B.S. Chlebus, and D.R. Kowalski, Time and communication efficient consensus for crash failures, in *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*, 2006, Springer LNCS 4167, pp. 314 - 328.
[12] B.S. Chlebus, and D.R. Kowalski, Locally scalable randomized consensus for synchronous crash failures, in *Proceedings of the 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2009, pp. 290 - 29.
[13] B.S. Chlebus, D.R. Kowalski, and J. Olkowski, Fast agreement in networks with byzantine nodes, in *Proceedings of the 34th International Symposium on Distributed Computing (DISC)*, 2020, pp 30:1–30:18.
[14] B.S. Chlebus, D.R. Kowalski, and M. Strojnowski, Fast scalable deterministic consensus for crash failures, in *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC)*, 2009, pp. 111 - 120.
[15] B. Chor, M. Merritt, and D.B. Shmoys, Simple constant-time consensus protocols in realistic failure models, *J. ACM*, 36(3):591–614, 1989.
[16] D. Dolev, and R. Reischuk, Bounds on information exchange for Byzantine agreement, *Journal of the ACM*, 32 (1985) 191 - 204.
[17] C. Dwork, J. Halpern, and O. Waarts, Performing work efficiently in the presence of faults, *SIAM Journal on Computing*, 27 (1998) 1457 - 1491.
[18] M. Fisher, and N. Lynch, A lower bound for the time to assure interactive consistency, *Information Processing Letters*, 14 (1982) 183 - 186.
[19] M. Fisher, N. Lynch, and M. Paterson, Impossibility of distributed consensus with one faulty process, *Journal of the ACM*, 32 (1985) 374 - 382.
[20] Z. Galil, A. Mayer, and M. Yung, Resolving message complexity of Byzantine agreement and beyond, in *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1995, pp. 724 - 733.
[21] C. Georgiou, D.R. Kowalski, and A.A. Shvartsman, Efficient gossip and robust distributed computation, *Theoretical Computer Science*, 347 (2005) 130 - 166.
[22] S. Gilbert, and D.R. Kowalski, Distributed agreement with optimal communication complexity, in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, pp. 965 - 977.
[23] I. Gupta, R. van Renesse, and K.P. Birman, Scalable Fault-Tolerant Aggregation in Large Process Groups, in *Proc., International Conference on Dependable Systems and Networks (DSN)*, (2001) 433 - 442.
[24] V. Hadzilacos, and J.Y. Halpern, Message-optimal protocols for Byzantine agreement, *Mathematical Systems Theory*, 26 (1993) 41 - 102.
[25] V. Hadzilacos, and S. Toueg, Fault-tolerant broadcast and related problems, in *Distributed Systems*, 2nd edition, Eddison-Wesley, 1993, pp. 97 - 145.
[26] M.T. Hajiaghayi, D.R. Kowalski, and J. Olkowski, Improved Communication Complexity of Fault-Tolerant Consensus, *CoRR abs/2203.12912* (2022).
[27] D.R. Kowalski, and J. Mirek, On the Complexity of Fault-Tolerant Consensus, in Proceedings of NETYS, 2019, pp. 19 - 31.
[28] D.R. Kowalski, and M. Strojnowski, On the communication surplus incurred by faulty processors, in *Proc., 21st Int. Sym. on Distributed Computing (DISC)*, 2007, pp. 328 - 342.
[29] Valerie King, Jared Saia Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary, J. ACM, (2011) 58, 18:1–18:24.
[30] D. Peleg, (2000). Distributed Computing: A Locality-Sensitive Approach. SIAM.
[31] M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, *Journal of the ACM*, 27 (1980) 228 - 234.