Nearly-Optimal Consensus Tolerating Adaptive Omissions: Why is a Lot of Randomness Needed?

Mohammad T. Hajiaghayi
University of Maryland, Maryland, USA.

Dariusz R. Kowalski
School of Computer and Cyber Sciences, Augusta University, Georgia, USA.

Jan Olkowski
University of Maryland, Maryland, USA.

Abstract

We study the complexity of the problem of reaching agreement in a synchronous distributed system, also called consensus, by n autonomous parties, when the communication links from/to faulty parties can omit messages. The faulty parties are selected and controlled by an adaptive, full-information, computationally unbounded adversary. We design a randomized algorithm that works in $O\left(\sqrt{n}\log^2 n\right)$ rounds and sends $O\left(n^2\log^3 n\right)$ total number of communication bits, where the number of faulty parties can be $\Theta(n)$. When the number of faulty parties is linear in n, our result is simultaneously tight for both these measures within polylogarithmic factors: due to the $\Omega(n^2)$ lower bound on the number of messages send by any Monte Carlo solution, by Abraham et al. (PODC'19), and due to the $\Omega(\sqrt{n/\log n})$ lower bound on the number of rounds of any Las Vegas solution by Bar-Joseph and Ben-Or (PODC'98). Thereby, this work settles the landscape of the consensus problem in the omission failures model, which stood as an open question since the work of Dolev and Strong (SICOMP'83).

Additionally, we strictly quantify how much randomness is necessary and sufficient to reduce time complexity to a certain value, while keeping the communication complexity optimal wrt to polylogarithmic factors. We prove that no Monte Carlo algorithm can work in less than $\Omega\left(\frac{n^2}{\max\{R,n\}\log n}\right)$ rounds if it uses less than $O(R)$ calls to a random source, assuming a constant fraction of all parties is faulty. This result should be contrasted with a long line of work on consensus algorithms against an *adversary limited to polynomial computation time*, thus unable to break cryptographic primitives, culminating in a work by Ghinea et al. (EUROCRYPT'22), where an optimal $O(r)$-round solution reaching consensus with probability $1-(cr)^{-r}$ is given. Our lower bound strictly separates these two regimes, by excluding such results if the adversary is computationally unbounded.

On the upper bound side, we show that for $R \in \tilde{\mathcal{O}}\left(n^{3/2}\right)$ there exists a randomized algorithm solving consensus in $\tilde{\mathcal{O}}\left(\frac{n^2}{R}\right)$ rounds, with probability polynomially close to 1 (whp), where tilde notation hides a poly-logarithmic factor. The communication complexity of the algorithm does not depend on the amount of randomness R and stays (universally) optimal within polylogarithmic factors. As a consequence, we give a spectrum of solutions that interpolates between optimal results in the deterministic regime ($R \in O(n)$; $O(1)$ entropy per party) and the randomized regime ($R \in O(n^{3/2})$ random bits).

1 Introduction

In any distributed system, reaching agreement (consensus) is essential for coordinating actions of the participating n parties (also called processes), out of which up to t could be faulty. The hardness of the task primarily depends on the type of fault present in the system. The classical hierarchies [9, 28] for synchronous message-passing models distinguish the following types of faults, in the order of increasing hardness: crash failures, omission failures, authenticated Byzantine failures, and Byzantine failures. In this work, we focus on the second type of failures in this list, the *omission failures*. Precisely, we assume that a computationally unbounded and malicious adversary can observe the system during the computation and *omit* an arbitrary subset of messages send to / received from selected faulty processes in an online, adaptive fashion. The adversary can also, based on the history of the computation, corrupt new processes if the number of corrupted stays within a fixed limit t. The adversary, however, cannot see the future random bits.

Despite of huge volume of research on the performance of consensus algorithms, the proper assessment of the hardness of consensus under this type of failure has been elusive. There is no theoretical evidence that omissions failures are weaker than the next model in the hierarchy, authenticated Byzantine, even if such a hypothesis seems compelling. Compared to the weaker model of crash failures, there is a strict hardness barrier following from the existence of an algorithm using $O(n^{3/2}\log^{13/2} n)$ messages by Hajiaghayi et al. [23] (STOC'22) in the case of crash failures and the $\Omega(n^2)$ lower bound on the number of messages in the model with omission failures, proved by Dolev and Reishuk [14] (JACM'85) for deterministic solutions and by Abraham et al. [1] (PODC'19) for the randomized ones, all results assuming $\Theta(n)$ faulty parties. However, if the space of solutions is categorized based on the round complexity of a solution, even for these two models (i.e., crashes and omissions) the picture is not clear. Both models admit an $\Omega(\sqrt{n/\log n})$ lower bound for Las Vegas solutions, due to the work of Bar-Joseph and Ben-Or [10] (PODC'98) (when the number of faulty parties is linear in the system size). In the case of crashes, the lower bound is matched by an algorithm of the same authors [10], while in the case of omissions – the best previous solution is 40 years old result of Dolev and Strong [15] (SICOMP'83) that works in $O(n)$ rounds! Hence, our first goal is to fully understand the hardness of omission failures in reaching consensus:
Question 1: Is there a consensus algorithm, possibly randomized, that at the same time matches both lower bounds with respect to poly-logarithmic factor? That is, is there an algorithm that solves consensus in $\tilde{O}(\sqrt{n})$ rounds with $\tilde{O}(n^2)$ total number of sent communication bits even if a linear number of omission failures occur in the network?

We answer this question affirmatively and give a new algorithm that is almost-optimal[1] with respect to, simultaneously, the number of rounds ***and*** the total number of sent communication bits when the number of faults is $\Theta(n)$.

Then, we focus on the aspect of how much randomness is necessary to break the $\Omega(n)$ lower bound on the number of rounds for deterministic solutions [17] (again, assuming $\Theta(n)$ faults):
Question 2: What is the impact of the amount of randomness available at processes on the efficiency of consensus algorithms? In particular, could pseudo-random generators be used efficiently?

We quantify the number of random bits that is sufficient and necessary to achieve a given time $\in \tilde{\Omega}(\sqrt{n})$[2], while keeping almost-optimal communication complexity. In particular, we prove that using pseudo-random generators with small seeds may delay reaching consensus nearly quadratically, which may have severe consequences in distributed ledger implementations and distributed database applications based on consensus. Even more interestingly, our lower bounds apply to

[1] In this work, almost-optimal means optimal within a poly-logarithmic factor.
[2] We use tilde notation to hide a polylogarithmic factor.

	result	time	comm. bits	random bits	comments
algo-	Thm 1	$O\left(\sqrt{n}\log^2 n\right)$	$O(n^2\log^3 n)$	$O\left(n^{3/2}\log^2 n\right)$	
rithms	Thm 3	$O\left(\frac{n^2}{R}\log^2 n\right)$	$O\left(n^2\log^5 n\right)$	$O\left(R\log^2 n\right)$	$\forall R \in O\left(n^{3/2}\right)$
lower	[10]	$\Omega\left(\frac{t}{\sqrt{n}\log n}\right)$	-	-	correct prob. $= 1$
bounds	[1]	-	$\Omega\left(\epsilon t^2\right)$	-	correct prob. $\geq \frac{3}{4}+\epsilon$
	Thm 2	T	-	R	$T\times(R+T)=\Omega\left(\frac{t^2}{\log n}\right)$
					correct prob. $\geq 1-\frac{1}{n^{3/2}}$

Table 1: Our main results presented for three metrics: time complexity, total number of used communication bits and total number of used random bits. All our algorithms are subject to their complexity bounds whp, i.e., with probability polynomially close to 1, see Section 2. T and R are random variables denoting, respectively, the number of rounds and the cumulative number of random bits used by all processes in a run of an algorithm. The "correct prob." denotes the probability of correctness of the class of algorithms to which the lower bound applies. The new results are underlined in column "result". Extended versions of Theorems 1 and 3, with explicit dependency on the parameter t, are given in Sections B and D, resp.

Monte Carlo algorithms. This makes a surprising distinction between a parallel line of work on consensus protocols in the case of authenticated Byzantine failures governed by a *computationally-bounded* adversary [2, 19, 21, 25], done by the cryptography community. In this model, the adversary is still adaptive, but the algorithm has a trusted setup for unique threshold signatures and an unforgeable public-key infrastructure; thus, the view of the adversary is limited to the history of the faulty processes and all messages send to them. Nevertheless, the adversary can adaptively corrupt new parties based on its view and has the priority over messages being delivered in the round it corrupts. In this setting, the result of Ghinea et al. [21] (EUROCRYPT'22) shows an algorithm that terminates in r rounds with probability at least $1-cr^{-r}$, for any r and some absolute constant $c>0$. Our lower bound excludes such results in the model with a *computationally unbounded adversary*, showing that not only many random bits but also strong cryptography could be necessary for some applications of consensus protocols.

Summary of results and the paper structure. As mentioned earlier, we focus on the classical message-passing model in which processes are autonomous, synchronous and can exchange messages via point-to-point communication channels. The adversary corrupts and controls at most t processes, causes omission failures at them, and is (full-information) strongly adaptive. In Sections 3, 4 and 5, we present an extended overview of the three main results and novel techniques obtained in this work; for the sake of overview, we typically assume $t=\Theta(n)$. A summary of them can be found in Table 1. In Section A we discuss the related work in more details. Section 2 presents detail formal description of the model. Sections B, C and D contain the full and formal analysis of our main results. Section 6 states major research directions opened by our work.

2 Model Details and Definitions

We consider the classical synchronous message-passing distributed system with omission faults, cf., [9, 28, 33]. The system contains n processes, also called parties. Each process has a unique ID in $\mathcal{P}=[n]=\{1,\ldots,n\}$. For simplicity, we will use p to refer to a process with ID $p\in\mathcal{P}$. Both $\mathcal{P}$ and n are known to all process. Processes operate in synchronized rounds. Without loss

of generality,[3] we assume that each round consists of the following two phases:

1. **Local computation phase**: Each process performs a local computation (i.e., autonomously from other processes) in order to change its state. The computation can be any function of the current state, of all messages received prior to this phase, and of a sequence of uniformly distributed and independent random bits of an arbitrary, but finite, length that can be reached by a process at the beginning of the phase. More precisely, for the last parameter of the function, we assume that there exists a random source that, when called, can provide a process and its state-changing function with a 0-1 sequence, of requested length, containing uniform and independent distributed random bits.

2. **Communication phase**: During this phase, each process can send messages to any other processes in the system. The content of a message is a function of the current state computed in the preceding local computation phase, and is not limited by the model. In particular, the length and content of messages is not restricted by the model, although our algorithms are designed in a way to use short messages. Each message sent is delivered to its destination at the end of the same phase, and is ready to be processed in the next round, unless an omission failure occurs.

Processes' omission failures and adversaries. Not all processes are reliable. Up to some $t = O(n)$ processes may become *(omission)* faulty during the execution – once a process becomes faulty, it stays faulty through the end of computation and some of its incoming/outgoing messages could be lost. We assume that t is a part of the problem input and thus known up-front to all processes. The decision which process becomes faulty and when, as well as the control over the faulty processes, is governed by an adversary. We consider an adaptive full-power full-information adversary that

- has unlimited computational power, knows the algorithm and input parameters and can see the states (and thus also the current random bits used) of all processes, as well as the content of all arriving messages, at any time, and

- can select online which (non-faulty) processes to fail and when, and with respect to faulty processes – it can *omit* any subset of messages incoming/outgoing to/from the faulty processes (i.e., such messages are not delivered to their destinations, having the same effect as no message sent).

Consequently, an *adversarial strategy* is a deterministic function, which assigns to each possible history that may occur in any execution some adversarial action for the subsequent *phase* of the execution, i.e., which processes to fail in that phase and in which moment and which messages sent by/to them would reach their destinations. Note, that in the above definition, the adversary has flexibility to adapt its actions between any two phases of the algorithm (not only between consecutive rounds). In the remainder, we will be referring to this adversary simply by *adaptive adversary*.

We remark that crash failures of processes can be viewed as omission failures – the adversary simply bans *all* their incoming and outgoing messages after the failure round (while in the round of a crash, the adversary could allow any subset of outgoing messages to reach their destinations).[4]

[3] See [9, 10] for the discussion on generality of our assumptions and related settings – crash and Byzantine faults.

[4] In case of crashes, incoming messages are not relevant, because the faulty process could not influence correct processes any more and it is not required from them to satisfy any of the consensus properties.

Consensus problem. A randomized algorithm for processes $\mathcal{P} = \{1, \ldots, n\}$, where each process $p \in \mathcal{P}$ holds initial input $b_p \in \{0, 1\}$, is a consensus protocol tolerating t faulty processes if all the three following conditions hold with probability 1 in the presence of an adaptive adversary failing/controlling at most t processes:

Agreement. All non-faulty processes output the same value.

Validity. If all non-faulty processes begin with the same input value b, then all of them output b.

Termination. Non-faulty processes have to decide and terminate.

Consider a randomized consensus algorithm against a fixed adversarial strategy. The following metrics determine the quality of the execution against that strategy:

- *Time of an execution* of the algorithm is defined as the smallest number τ_1 such that the number of rounds that occur by termination of *the last non-faulty process* is at most τ_1;
- *The number of communication bits in an execution* of the algorithm is the smallest number τ_2 such that the total number of bits sent by *all processes* in point-to-point messages by termination of the last non-faulty process is at most τ_2;
- *Randomness of an execution* of the algorithm is defined as the smallest number τ_3 such that the number of (independent and uniform) random bits used by *all processes* by termination of the last non-faulty process is at most τ_3; when describing the lower bound result, we abuse the notation slightly, and define randomness of an execution as the total number of times when processes access their random sources during the local computation phase. Complying to the definition of the local computation phase, in each such access a process can use a sequence of random bits of finite length. Observe that such definition makes a lower bound result even stronger.

We define *time/communication/randomness complexity of a (distributed) algorithm* as a supremum of time, the number of communication bits, and the number of random bits, respectively, taken over all adversarial strategies.[5] Finally, *time/communication/randomness complexity of a distributed problem* is an infimum of all algorithms' time/communication/randomness complexities, respectively. In our paper, we are interested in studying almost-optimal solutions, i.e., optimal within a polylogarithmic factor, wrt the abovementioned complexities. The bounds on the complexities should hold with high probability. We say that a random event occurs *with high probability* (*whp* for short), if its probability could be made $1 - O(n^{-c})$ for any positive constant c by linear scaling of parameters of the considered random process.[6]

3 Main consensus algorithm

Our first and main result is a new consensus algorithm which is almost-optimal with respect to time complexity, bit complexity and randomness complexity, if the number of faulty parties is $\Theta(n)$.

[5] Note that the supremum for different measures could be achieved by sequences of different strategies – nevertheless, each of the complexities of our solutions is still close to optimal.

[6] We would like to note that, after ignoring polylogarithimic factors, all our upper bounds hold with the same asymptotic complexities in even stronger regime where the probabilities are of form $1 - O(n^{\omega(1)})$.

Algorithm 1: OPTIMALOMISSIONSCONSENSUS

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

$\texttt{operative}_p \leftarrow true$, $\texttt{decided}_p \leftarrow false$;

$V_p \leftarrow$ a set of neighbors of p in a predetermined graph G guaranteed by Theorem 4;

$W_1, \ldots, W_{\lceil\sqrt{n}\rceil} \leftarrow$ a pre-defined partition of $\mathcal{P}$ into $\lceil\sqrt{n}\rceil$ disjoint sets of size $\leq \lceil\sqrt{n}\rceil$ each;

let ℓ be such that $p \in W_\ell$;

for $\frac{t}{\sqrt{n}} \log n$ *epochs* **do**

  $\texttt{g\_ones}_p, \texttt{g\_zeros}_p, \texttt{operative}_p \leftarrow$ GROUPBITSAGGREGATION$(W_\ell, p, \texttt{operative}_p; b_p)$;

  if $\textit{operative}_p = false$ **then** stay idle until the end of the epoch;

  $\texttt{ones}_p, \texttt{zeros}_p, \texttt{operative}_p \leftarrow$ GROUPBITSSPREADING$(V_p, p, \ell, \texttt{operative}_p; \texttt{g_ones}_p, \texttt{g_zeros}_p)$;

  if $\textit{ones}_p > \frac{18}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $b_p \leftarrow 1$;

  else if $\textit{ones}_p < \frac{15}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $b_p \leftarrow 0$;

  else set b_p to 0 or 1 uniformly at random;

  if $\textit{ones}_p > \frac{27}{30}(\textit{ones}_p + \textit{zeros}_p)$ *or* $\textit{ones}_p < \frac{3}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $\texttt{decided}_p \leftarrow true$;

end

if $\textit{operative}_p = true$ *and* $\textit{decided}_p = true$ **then send** b_p to all processes in $\mathcal{P}$;

else if *any message* b_q *received from some process* q **then** $b_p \leftarrow b_q$;

`/* in the above,` q `can be chosen arbitrarily from the received messages */`

if $\textit{decided}_p = true$ *or* ($\textit{operative}_p = false$ *and* p *received a message in the previous round*) **then decide** b_p;

else

  if $\textit{operative}_p = true$ **then** p participates in the deterministic synchronous Consensus algorithm given in Theorem 4 in [15] with the input bit b_p; if p reaches agreement in that protocol, it broadcasts the decision to all processes in $\mathcal{P}$ and it decides on the algorithm's decision;

  else p remains idle until a decision is sent to it; upon receiving a decision, it decides on this value;

end

Theorem 1. *There is a randomized algorithm solving consensus with probability* 1 *against the adaptive omission adversary that can control* $t < \frac{n}{30}$ *processes, which terminates in* $O\left(\sqrt{n}\log^2 n\right)$ *rounds and uses* $O\left(n^2 \log^3 n\right)$ *bits of communication and* $O\left(n \cdot \sqrt{n}\log^2 n\right)$ *random bits, whp.*

For the ease of presentation, and for the sake of space constraints, in this section, we assume that $t = \frac{n}{30} - 1$ and we provide a more high-level overview of techniques used to obtain the result. A self-contained and fully formal derivation of this theorem, incorporating the upper bound t on the number of faulty processes into the time and communication complexities and containing all the omitted proofs, is deferred to Appendix B, where the above Theorem is restated as Theorem 5.

In the case when $t = \Theta(n)$, the almost-optimality of the running time follows from the $\Omega\left(\sqrt{n/\log n}\right)$ lower bound showed in [10]. The almost-optimality of the communication bit complexity is due to the result of Abraham et al. [1] who showed that any randomized algorithm solving consensus with at least a constant positive probability against the adaptive omission-causing adversary requires $\Omega(n^2)$ messages (each message carries at least one bit). The almost-optimality of the randomness complexity follows from Theorem 2 presented in the later part of the paper. We next give an overview of the algorithm. The pseudocode can be found in Algorithm 1.

Universal idea: Local and dynamic partitioning of processes into operative / inoperative and implementing time- and communication-efficient biased-majority-voting only by the operative ones. We introduce a new partitioning of processes into operative and inoperative, based on the communication received by each process from a certain pre-defined set of other processes which maintain their operative status (this set may vary, depending on what procedure is executed – it will be emphasized later). This partition is not equivalent to the standard classification into faulty / non-faulty ones. With our partition, we can guarantee that faulty processes either communicate well enough to contribute to the progress towards a unified decision (i.e., stay operative) or become excluded from the set of operative processes, having no impact on the final decision. Our partition also avoid a major performance problem in omission-tolerant or Byzantine-tolerant computation – identifying a single faulty process may require at least quadratic number of messages, cf., [1], which makes it fast, local and incorporated in efficient communication schedules.

Then, we employ the idea of reaching consensus by applying the biased-majority-voting rule, as proposed in [10], but with a novel twist – only the operative processes implement the vote protocol to agree on a consensus decision. It uses $O(\sqrt{n}\log n)$ repetitions of the single vote subroutine, called *epochs* in the pseudo-code, see lines 6-12 in Algorithm 1. Each repetition/epoch consists of $O(\log n)$ rounds of communication-efficient counting, see the description below), however it succeeds in unifying the votes only if the number of newly failed processes is $O(\sqrt{n})$ and only with a constant probability (this is why we need $O(\sqrt{n}\log n)$ epochs). Only after the part implementing the biased-majority-voting rule ends, the operative processes communicate the decision to the remaining parties. We first describe how we implement a single epoch, based on two technical advancements, and conclude with more details on how the overall consensus protocol (based on the biased-majority-voting rule) is designed.

An implementation of a single biased-majority-vote subroutine (epoch). In our case, a single epoch (i.e., a single repetition of the biased-majority-vote subroutine, lines 6-12) heavily relies on **counting**, collaboratively by every operative process, the number of operative processes that have candidate decision value 0 and, separately, value 1. These numbers must be approximate, up to an additive factor linearly dependent on the number of processes that become inoperative, as the operative status may change dynamically – some processes can lose it before the calculation finishes and, in consequence, their candidate values might not be properly counted by others. Moreover, this calculation has to consider the fact that some operative processes can be controlled by adversary, thus it must, regardless, exploit the property that the operative processes communicate with enough other processes. A protocol performing this calculation in $O(\log n)$ rounds and using $O(n^{3/2}\log^2 n)$ communication bits, in total, is the main technical advancement of this algorithm and we present it next in a form of ***two technical advancements.***

Technical advancement 1: $\sqrt{n}$-decomposition into groups and binary-tree-like intra-group calculations of operative processes for communication saving. As mentioned above, there are some inherit difficulties in the omission failure model that complicate the time- and communication-efficient counting of the number of candidate values (i.e., votes) 0 and 1 among the operative processes. Here, we present the techniques we use to mitigate them. We pre-define fixed partition of the set of processes into $\lceil\sqrt{n}\rceil$ *groups* of size $\lfloor\sqrt{n}\rfloor$ or $\lceil\sqrt{n}\rceil$ each (see line 3 of Algorithm 1 and example in Figure 1), and first require the operative processes to count the number of operative 0's and 1's only within the groups (executed procedure GroupBitsAggregation in line 6). In this part, we use a virtual sparse data structure on some subsets of processes, structured into a balanced binary tree, which is used to aggregate the counts.

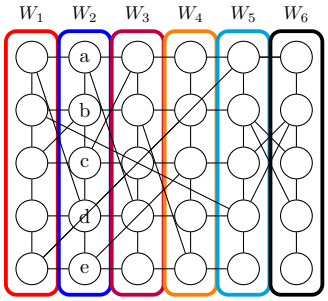

Figure 1: A schematic picture of two different techniques used for communication between processes. Different colors represent different groups in the $\sqrt{n}$-decomposition of the processes. The links represent the overlaying communication resembling a sparse random graph used for exchanging operative counts of different groups. The choice of links is independent of the $\sqrt{n}$-decomposition.

Vertices of the binary tree correspond to specific subsets of processes in the group: leaves of the binary tree are singletons in the group, and each vertex in a higher layer corresponds to the union of the subsets that are already identified with the children of that vertex in the tree. The root of the binary tree corresponds to the set of all processes in the group. Processes calculate the number of operative 0's and 1's, called *operative counts*, starting from the leaves of the tree (i.e., singletons) and then keep moving up the tree. At each vertex in a higher layer, the processes in the subset corresponding to that vertex work together to relay and sum up the operative counts from the lower layer. In this relay-and-aggregation procedure, all operative processes in the group exchange messages, not only to relay and aggregate values from children to parents in the virtual tree, but also to keep track who remains operative in the whole group. More precisely, if a process receives information from less than half of the other processes from the group, during the procedure of relaying and aggregating the operative counts of 0's and 1's from the lower layer, it becomes inoperative. See example in Figure 2. After $O(\log n)$ rounds, corresponding to the height of the binary tree, the operative counts for the entire group – corresponding to the root of the tree – can be calculated. Complying with the rule of receiving enough number of messages in order to maintain an operative status, only the processes of the group that remain operative use the operative counts further in the protocol. In the analysis, we will be able to prove that, regardless of the omission failures' pattern, there is a group of $\Theta(n)$ operative processes whose counts of operative 0's and 1's differ by, at most, the number of processes who have become inoperative. Taking the advantage of the binary-tree-structured communication, we can guarantee that processes of each group exchange at most $\tilde{O}(n)$ bits in total and that the procedure of operative counting 0's and 1's takes only $O(\log n)$ rounds. Summarizing, we prove the following result about a single execution of the procedure GroupBitsAggregation. Detailed description can be found in Appendix B.1 and in Algorithm 2; the formal analysis is in Appendix B.2.

Lemma (Lemma 1 and 2 in Appendix B.2)**.** *A single execution of the procedure* GroupBitsSpreading *works in $O(\log n)$ rounds and guarantees that every operative process in a single group knows an approximate number of other operative processes in the group having candidate value 0 and 1. The numbers in different operative processes differ by at most the number of processes of the group that became inoperative during the execution of the procedure. Processes in a single group use at most $O(n \log^2 n)$ bits of communication, in total, during the execution.*

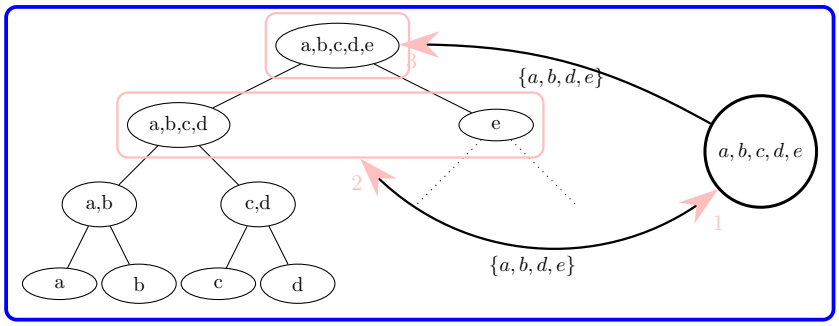

Figure 2: Visualization of the $\sqrt{n}$-decomposition of the blue group from Figure 1. The processes a, b, c, d, e in the group are logically decomposed into a binary tree. The pink arrows visualize the three-round process of relaying operative counts of the two children of the root to the root itself. First, the counts are relayed to all processes in the group (arrow #1), then the processes send a confirmation if they received the counts (arrow #2), finally, all in the group transmit the received counts to the higher layer – the root in this case (arrow #3). Some processes can be faulty (process c does not communicate, only $\{a, b, d, e\}$) and their values are not guaranteed to be accumulated accurately.

Technical advancement 2: Fast inter-group communication and status maintenance between operative processes. After the tree-based communication, the operative processes in each group have a shared knowledge about the count of operative 0's and 1's within their group. Since there are $\lceil\sqrt{n}\rceil$ different groups, the number of logically different counts is $O(\sqrt{n})$. To exchange these $O(\sqrt{n})$ counts between the groups, the operative processes communicate along the links in a sparse, but well-connected, graph – neighborhoods of which are pre-selected locally in line 2 – that underlays the entire network; see the executed procedure GroupBitsSpreading in line 8 of Algorithm 1 and the illustration of the graph on the top of the group partition in Figure 1. The graph used by the operative processes is selected as follows. We consider a random graph, where each edge is selected independently at random with probability $\Theta(\log n/n)$. Next, we use the probabilistic analysis to show that the following event holds whp: a random graph with such edge density has the property that every subgraph of a constant-fraction size is dense and shallow – see Theorem 4 in Appendix B.2. The "dense" property refers to the fact that removing an arbitrary but at most α-fraction of edges incident to any vertex from a linear number of vertices, allows to find a connected subgraph of this linear number of vertices such that every vertex has degree at least $\beta \log n$ within this subgraph. The "shallow" property refers to the fact that the latter subgraph has logarithmic diameter (asymptotically). These two properties justify our partition into operative and inoperative classes, from perspective of the inter-group communication: as long as the process has more than $\beta \log n$ active links, intuitively it belongs to the connected shallow subgraph and thus it is capable of exchanging information with any other process with this property in $O(\log n)$ rounds. This holds regardless of the factual faulty / non-faulty state of the processes. Therefore, the operative processes can spread among themselves the operative counts of the $\lceil\sqrt{n}\rceil$ different groups, yielding the property that as long as a process remains operative it knows *some operative counts of any group* with at least one operative process. More specifically, every operative process stores a data structure memorizing the operative counts of each of the $\lceil\sqrt{n}\rceil$ groups present in the system. Initially, it knows only the counts of the group it belongs to. In $\Theta(\log n)$ rounds of communication it keeps sending these counts along every edge determined by the underlying graph, maintaining the fact that operative counts of a particular group are sent only once via each

edge; at the same time it receives counts of other groups and updates the data structure based on this information. In case a process receives two or more different count values of some group, it can choose arbitrarily any of them – as argued earlier, all of them could differ by at most the number of processes that have become inoperative in that group. To summarize, the procedure GroupBitsSpreading has the following outcome (the formal description is given in Algorithm 3 and in Appendix B.1; the formal analysis is provided in Appendix B.2).

Lemma (Lemmas 6, 8 and Theorem 5 in Appendix B.2)**.** *Assume that processes run the procedure* GroupBitSpreading *with $O(\sqrt{n})$ different logical input values (which are the operative counts of candidate values* 0 *and* 1 *of each group). At the end of the procedure, each operative process knows at least one copy of the logical value, provided that at least one process starting with this logical value remains operative. The procedure uses $O(\log n)$ communication rounds and $O(n\sqrt{n}\log^2 n)$ communication bits in total.*

The combination of the two above technical advancements lead to calculating the number of candidate values 0 and 1 among the operative processes. Although these numbers are approximate, as they can differ by the number of processes that have become inoperative, this difference is acceptable to still employ a variant of the biased-majority-voting consensus, as we discuss in the next part.

Putting them all together: consensus protocol based on biased-majority-voting adjusted to the new efficient voting implementation in an epoch. Assuming that each operative process has the approximate numbers of other operative processes having a candidate value 1 and 0, we explain the modification to the consensus framework based on the biased-majority-voting by Bar-Joseph and Ben-Or [10] in order to adapt to the dynamic characteristic of the operative set of processes and to the properties that our new efficient single-epoch implementation of a voting has. Our modification takes into account that, in our case, the counts of operative 0's and 1's are not the same in every operative process. The communication protocols guarantee only that a candidate value of an operative process is accounted by any other operative process, however, it gives no guarantee regarding the candidate values of the processes that become inoperative during the calculation and communication. Also, in our implementation of a single biased-majority-voting subroutine (epoch), described earlier, the operative processes do not assign any default values to the candidate values of the inoperative processes. Thus, any operative process estimates the number of all operative processes simply by adding the operative counts of 0's and 1's. Based on the above estimates, the algorithm employs the procedure of converging the candidate values of the operative processes to the final decision value according to the following: if an operative process has an operative count of 1's (meaning the candidate values of operative processes assigned to 1) at least $\frac{18}{30}$ of the estimated total of all operative processes, it sets the candidate value to 1. If the operative count is less than half of the estimated total, it sets the candidate value to 0. In all other cases, the candidate value for the next step is a uniformly chosen random bit. See lines 9-11 and Figure 3 for an illustration.

Following the main line of the biased-majority-voting idea, in the analysis, we show that if a perturbation in the number of operative processes before the counting starts and after it ends is small, i.e., $O(\sqrt{n})$, the estimations are good enough to reach a consensus decision by operative processes, with constant probability i.e. the candidates values among operative processes are unified. A high level intuition is that if there is a fraction of 1's as the candidate values among the operative processes of at least $\frac{18}{30}$ then every operative processes assigns 1 as its candidate value. Otherwise every operative process either sets a random bit or 0 as the candidate value, however, the standard

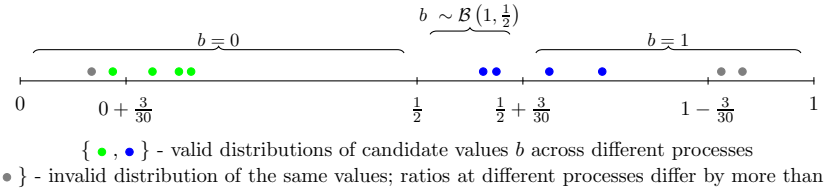

Figure 3: A picture explaining the thresholds in a single execution of the biased-majority-voting subroutine, see lines 9-11 in Algorithm 1. Different colors represent different outcomes (each obtained in a different epoch) of the counting of candidate values in preceding lines 6 and 8.

bounds on the deviation from the mean of the sum of many i.i.d. random variables guarantee that, with constant probability, the number of assigned 0's is below the mean by $\Theta(\sqrt{n})$. If the perturbation is indeed $O(\sqrt{n})$, which occurs with a constant probability, this leads to assigning 0 as the candidate value in the next repetition of the biased-majority-voting subroutine by every operative process. Since the convergence analysis of a similar procedure has been done earlier in [10], but with different constants, we omit the details here and refer the reader to Lemmas 10, 11 in Appendix B.2.

On the other hand, the communication graphs in the communication protocols are designed to guarantee at least $n - \Theta(t)$ operative processes in the system, regardless of the adversary's actions.

Lemma (Lemma 7 in Appendix B.2)**.** *The number of operative process is always at least* $n - 3t$.

Therefore, after $O\left(\frac{t}{\sqrt{n}} \log n\right) \leq O(\sqrt{n} \log n)$ epochs, each executing our biased-majority-voting subroutine, we can assert, by a counting argument, that there were at least $\Omega(\log n)$ epochs in which the perturbation to the number of operative processes was $O(\sqrt{n})$. In consequence, we can argue that operative processes have reached a consensus decision with high probability – the voting in different epochs use independent random bits, thus their outcome are independent and standard bounds on probability of success of $\Theta(\log n)$ independent trials of a Bernoulli variable can be applied. As a final step, the operative processes disseminate the decision to all processes by all-to-all-communication, see lines 14-16.[7]

The detailed analysis of Algorithm 1 is presented in Appendix B.2 and the precise formal result is stated in Theorem 5, however, we note here the complexity milestones. The communication bit complexity is always $O\left(\frac{t}{\sqrt{n}} \log n \cdot n^{3/2} \log^2 n + n^2 + nt\right) = O\left(n\left(t \log^3 n + n\right)\right)$. The first additive term corresponds to $O\left(\frac{t}{\sqrt{n}} \log n\right)$ repetitions of our biased-majority-voting subroutine (i.e., epochs), each requiring $O\left(n^{3/2} \log^2 n\right)$ bits of communication. The additive term $O\left(n^2\right)$ corresponds to the procedure of informing other processes of the decision made by the operative processes. The last term $O(nt)$ corresponds to the execution of the deterministic protocol. The number of random bits used is at most 1 per process per the repetition of our biased-majority-voting subroutine (i.e., once per epoch), which gives $O\left(t\sqrt{n} \log n\right) \leq O\left(n\sqrt{n} \log n\right)$ bits of randomness in

[7] To increase the probability of success to 1, after each epoch, the operative processes employ a safety rule: if the estimation of operative processes holding candidate value 1's constitutes at least $\frac{27}{30}$ fraction of the overall estimation of all operative processes (or, conversely, the estimation of 1's constitutes less than $\frac{3}{30}$ fraction), a process sets an auxiliary variable `decided` to *true*, see line 12. The undecided processes may eventually switch to a deterministic protocol (line 18), working in $O(n)$ rounds and sending $O(n^3)$ communication bits cf. [15], but the underlying idea is that, based on the previous arguments, it is only with a probability of less than $O\left(\frac{1}{n}\right)$ that an operative process remains undecided throughout the $O\left(\frac{t}{\sqrt{n}} \log n\right) \leq O(\sqrt{n} \log n)$ repetitions of our biased-majority-voting subroutine. As this part is highly technical, we postpone details to Section B.

total. Similarly, each repetition of an the counting scheme takes $O(\log n)$ rounds, which implies $O\left(\frac{t}{\sqrt{n}}\log^2 n\right) \leq O\left(\sqrt{n}\log^2 n\right)$ round complexity, conditioned on the fact that the deterministic consensus algorithm is not evoked – this happens whp. In the case where the deterministic algorithm is executed, it takes additional $O(t)$ rounds after which all non-faulty processes decide with probability 1, but this happens with polynomially small probability.

4 Lower bound

Our next result shows a new connection between randomness, which we informally define as the total number of accesses to a random source by the algorithm,[8] and time complexity in consensus solutions against an adaptive adversary. It provides a lower bound for a broader class of consensus algorithms that are correct with high probability,[9] against an adaptive adversary who can crash processes permanently – it automatically extends to omissions and more severe faults. Compared to previous lower bounds, we introduce a new, amortized analysis of the valency framework [10, 18, 31] – a common tool to deriving lower bounds for consensus algorithms. By crafting a new parameterized approach to the coin-flipping game – an abstraction that models processes' random choices – we can fully adapt the power of adversary to the amount of randomness the algorithm is using.

Theorem 2. *For a synchronous algorithm solving consensus with probability* $\geq 1 - \frac{1}{n^{3/2}}$*, let* T *denote the number of rounds in an execution of the algorithm and* R *be the total number of times the processes have accessed a random source. There exists an adaptive adversarial strategy that guarantees, with probability at least* $1 - \frac{1}{\log n}$*,*[10]

$$T \times (R + T) = \Omega\left(\frac{t^2}{\log n}\right) .$$

The above result is proved in Appendix C, as Theorem 7. It shows a *trade-off* between fast algorithms and algorithms that are frugal in their calls to a random source. Lower bounds of similar flavor have been already known for the asynchronous setting. Aspnes (JACM'98) [5] was the first to show that $\Omega\left(\frac{t^2}{\log^2 t}\right)$ coin flips are needed to solve asynchronous consensus. That result has been later improved by Attiya and Censor (STOC'07) [7], who obtained a tight bound showing that asynchronous consensus requires $\Theta\left(n^2\right)$ (asynchronous) step complexity. Note that the adversary in an asynchronous setting is much more powerful than in the synchronous one, considered in this paper, since he can delay *all* operations arbitrarily – we obtain similar lower bound (right-hand side of our formula) without being able to use an advantage of asynchrony in adversarial strategies.

In case of synchronous algorithms, the only known lower bound was obtained by Bar-Joseph and Ben-Or [10], who showed that no algorithm can solve consensus with probability 1 against the adaptive adversary in fewer than $\Omega(\frac{t}{\sqrt{n\log n}})$ rounds, even with unlimited randomness. Since a single process can have at most one call to a random source in a round, substituting $R := n \times T$ in our lower bound also implies that any algorithm solving consensus with probability at least $1 - \frac{1}{n^{3/2}}$ has to work for T rounds such that $T \times T \times (n+1) = \Omega(\frac{t^2}{\log n}) \implies T = \Omega(\frac{t}{\sqrt{n\log n}})$ with probability at least $1 - \frac{1}{\log n}$. Thus, we extend the bound in [10] to a wider class of algorithms

[8] Observe that this definition is, in principle, more general than counting only the total number of random bits. See Section 2 for formal specification of randomness.

[9] Algorithms correct with probability 1 are also in this class.

[10] In fact, for the cost of poly-logarithmic times more faults, the bound could be polynomialy close to one.

– Monte Carlo solutions. On the other hand, in the case of having a small (e.g., constant or polylogarithmic) number of processes who make random calls in a round, or when process uses pseudo-random generators of small (e.g., polylogarithmic) seed and the number of failures is big (e.g., $t = \Omega(\frac{n}{\text{polylog } n})$) – the lower bound implies $T = \Omega(\frac{t}{\text{polylog } n})$, which is (almost) equivalent to the renowned lower-bound $T = \Omega(t)$ (cf. [8]) for deterministic executions. Nevertheless, our lower bound describes precisely the relationship between the time and the number of random calls in the entire spectrum between the two extremes of unlimited randomness and determinism.

Lower bound's technical novelty and overview of its analysis. We propose a new improved analysis of the one-round coin-flipping game – an abstraction proposed by Bar-Joseph and Ben-Or [10]. They showed that, from a high level perspective, if n processes use randomness in a round, the adversary (knowing the random outcomes) can hide $\Theta(\sqrt{n \log n})$ values (which corresponds to failing $\Theta(\sqrt{n \log n})$ processes) such that with probability at least $1 - \frac{1}{n}$ the execution cannot be close to deciding. We improve this analysis and make it *parameterized* with respect to the number of calls to a random source, see Lemma 12 in Appendix C, in order to be able to *amortize* this number in the final analysis in the proof of Theorem 2 (equivalently, Theorem 7 in Appendix C).

More specifically, using Talagrand's concentration inequality, we show that even if only $k < n$ processes decide to make a call to a random source, the adversary can fail at most $\Theta(\sqrt{k \log \alpha})$ of them in such a way that the probability of preventing decision is at least $1 - 2^{\alpha}$, for $\alpha < \frac{1}{2}$. Introducing the artificial parameter α allows the adversary to control the game with almost any desired precision with the cost of only $\log \alpha$ times more failures in an execution.

Having this tool in hand, one could follow the generic framework of analyzing valency of the executions, as used by Bar-Joseph and Ben-Or in [10] and also in other related contexts in distributed computing, c.f., [5], [7]. In short, the framework relies on partitioning the executions into a finite and small number of exclusive types (also referred to as valency types) of executions that capture the probabilities of the algorithm deciding 0 or 1, given the history of the execution up to some point in time. More precisely, let $\Pr(\mathcal{H}, \mathcal{A})$ be the probability of reaching consensus on value 1 when continuing the run of the algorithm with history $\mathcal{H}$ under adversarial strategy $\mathcal{A}$. We say that a state of the algorithm in round i, defined uniquely by its history $\mathcal{H}$, is:

- *null-valent* if for all adversarial strategies $\mathcal{A}$ extending this state, we have $\frac{1}{n \log n} - \frac{i}{n^2} \leq \Pr(\mathcal{H}, \mathcal{A}) \leq 1 - \frac{1}{n \log n} + \frac{i}{n^2}$,
- 1-*valent* if there is an adversarial strategy $\mathcal{A}$ extending this state such that $\Pr(\mathcal{H}, \mathcal{A}) > 1 - \frac{1}{n \log n} + \frac{i}{n^2}$ and for every other adversarial strategy $\mathcal{A}'$: $\Pr(\mathcal{H}, \mathcal{A}') \geq \frac{1}{n \log n} - \frac{i}{n^2}$,
- 0-*valent* if there is an adversarial strategy $\mathcal{A}$ extending this state, such that $\Pr(\mathcal{H}, \mathcal{A}) < \frac{1}{n \log n} - \frac{i}{n^2}$ and for every other adversarial strategy $\mathcal{A}'$: $\Pr(\mathcal{H}, \mathcal{A}') \leq 1 - \frac{1}{n \log n} + \frac{i}{n^2}$,
- *bivalent* if there are adversarial strategies $\mathcal{A}, \mathcal{A}'$ extending this state, such that $\Pr(\mathcal{H}, \mathcal{A}) > 1 - \frac{1}{n \log n} + \frac{i}{n^2}$ and $\Pr(\mathcal{H}, \mathcal{A}') < \frac{1}{n \log n} - \frac{i}{n^2}$.

An execution that is 1-valent or 0-valent is also called *uni-valent*. Note that the types are disjoint and cover the whole space of an algorithm's states.

The classical approach is to show that (i) there exists an ambiguous assignment of input bits to processes with the probabilities of deciding 0 and deciding 1 being far from 1, and (ii) the adversary can keep the algorithm in the ambiguous state regardless of the algorithm's actions for a certain number of rounds. Step (i) is guaranteed by the following lemma:

Lemma (Lemma 13 in Appendix C)**.** *For any synchronous consensus algorithm there exists an initial state, which, if the adversary can control one process, is null-valent or bivalent.*

We now focus on Step (ii). In order to obtain amortized analysis of the number of calls to random sources and simultaneously to enforce the sought time lower bound, we give much tighter analysis of this type. In particular, we have to take into account (a) the number of accesses to random sources when analyzing transitions between different types of states, c.f., Lemmas 14 and 15, and (b) the amortized number of accesses when classifying states in round i according to their valency, c.f., the conditions defining the types of states based on valency (see above).

Lemma (Lemma 14 in Appendix C)**.** *A state $\mathcal{H}_i$ that is null-valent at the beginning of round $i < n$ can be extended to a null-valent state at the end of the round with probability greater than $1 - \frac{2}{n^2}$ by failing at most $16\sqrt{r_i \log n}$ processes.*

Lemma (Lemma 15 in Appendix C)**.** *Let $\mathcal{H}_i$ be a bivalent state. By failing at most $16\sqrt{r_i \log n}+1$ processes per round, the adversary can extend the state for the next $i' > 1$ rounds, with probability at least $1 - \frac{i'}{n \log n}$, reaching a state $\mathcal{H}_{i+i'}$ that is either bivalent or terminating. The latter case can happen only because failing the necessary processes in round $i + i' - 1$ would exceed the limit t on the total number of failures.*

Using the above lemmas, we can prove Theorem 2 (equivalent to Theorem 7 in Appendix C).

Proof of Theorem 2. By Lemma 13 in Appendix C (see also its statement above), the adversary can assign input values such that the initial state is in either a bivalent or a null-valent state. Then, the adversary follows the strategy described in Lemmas 14 and 15 in Appendix C (see also the statements above), depending whether the current state is null-valent or bivalent. Specifically, if the state is null-valent, the adversary can extend the execution by one more round with probability at least $1 - \frac{1}{n^2}$, again by Lemma 14 in Appendix C. If the state is bivalent, it can extend the state for some $i' > 1$ rounds with probability at least $1 - \frac{i'}{n \log n}$, such that the new state is again either bivalent or null-valent, or the execution terminates but then the number of failed processes in the previous round would exceed the adversary's limit t. If the algorithm decides to terminate, it must be in either a 0-valent or 1-valent state, since the algorithm is $\left(1 - \frac{1}{n^{3/2}}\right)$-strongly-correct. Therefore, the adversary can prolong the execution either for n rounds or until it runs out of the processes to fail.

Let T be the round in which the execution terminated. If $T = n$, then the theorem follows. Assume then that the adversary stopped implementing its strategy in round T due the fact that in the preceding round it could not fail the desired number of processes. Since the adversary fails at most $16\sqrt{r_i \log n} + 1$ processes in a round i (c.f., Lemmas 14 and 15 in Appendix C), we obtain

$$t \leq \sum_{i=1}^{T-1} \left(16\sqrt{r_i \log n} + 1\right) \leq 32 \sum_{i=1}^{T-1} \sqrt{(r_i + 1) \log n} \ ,$$

which is equivalent to

$$t^2 \leq 1024 \left(\sum_{i=1}^{T-1} \sqrt{r_i \log n}\right)^2 \ .$$

Applying the Cauchy-Schwarz inequality to the right hand side of the above, we get

$$t^2 \leq 1024 \left(\sum_{i=1}^{T-1} \sqrt{(r_i + 1) \log n}\right)^2 \leq 1024(T-1) \left(\sum_{i=1}^{T-1} (r_i + 1) \log n\right) \ ,$$

which, after proper rearranging, yields

$$\frac{t^2}{1024 \log n} \leq (T-1) \times (R+T)$$

and proves the theorem. □

5 Interpolation between random and deterministic solutions

To complement the lower bound from Theorem 2, we give a trade-off algorithm that matches it (with respect to poly-logarithmic factor) and each access to random source gets only one bit.

Theorem 3. *For any $R \in O(n^{3/2})$, there exists an algorithm solving consensus (with probability 1) against the most powerful, adaptive omission-causing adversary failing at most $t < \frac{n}{60}$ processes, which terminates in $\tilde{O}\left(\frac{n^2}{R}\right)$ rounds and uses, in total, $\tilde{O}\left(n^2\right)$ bits of communication and $\tilde{O}\left(R\right)$ bits of randomness, whp.*

The extended version of the above result is formally described and proved in Section D, as Theorem 8. We give here a general overview. The algorithm uses an idea of grouping processes into smaller subsets, first solving consensus within each subset and then propagating the decision to the entire system. Such a scheme is typically heterogeneous, in the sense that intra-group procedures/consensus mimics an efficient randomized algorithm while the inter-group procedures are based on efficient deterministic solution. Therefore, the smaller the groups the more randomness could be reduced (in theory). Grouping algorithms were first used in [23], where a similar trade-off between *communication/random-bit-complexity* and time was shown in case the processes are prone to crashes only (more benign faults). However, the same framework cannot be used here because, as discussed earlier, omission failures automatically introduce at least quadratic communication complexity, while the scheme in [23] relied on lower-communication procedures.

Therefore, our omission-based implementation is very different from the crash-based one in [23] and relies, once more, on the idea of dividing processes into operative and inoperative and on combining this approach with a round-robin algorithm working on groups (i.e., considering groups one-by-one). For the sake of clarity, assume that we split the set of processes $\mathcal{P}$ into x groups $SP_1, \ldots, SP_x$, of size at most $\lceil \frac{n}{x} \rceil$ each. In x subsequent phases, we let each group invoke our almost-optimal consensus algorithm against omissions (limited only to the members of the group), given by Theorem 1, one group per phase. Then, we propagate the decision of the consensus algorithm to other processes in the system and require that each subsequent call to the almost-optimal consensus algorithm, which takes place in any later phase, uses the propagated value as the input bit for the processes involved in the call in this phase.

The crux is in executing the above only among operative processes. Similarly to the algorithm presented by Theorem 1, we use a result about random graphs of expected degree $\Theta(\log n)$ for specifying the communication graph between groups. We also determine whether a process is operative or not based on the number of messages it received in the random graph. Ultimately, we again use the observation that the operative processes, ergo those who constantly receive many messages from processes who are their neighbors in the communication graph, can exchange any information between themselves within $O(\log n)$ rounds. This property is crucial for correctness of our algorithm and its detailed explanation can be found in the proof of Lemma 8. It then follows that if a group, which has sufficiently many non-faulty and operative processes, calculated a decision value, this value is properly distributed to all other operative processes. From now

on, the operative processes have the same input value in all remaining phases and, therefore, the adversary cannot prevent them from deciding on that value. The existence of a group satisfying the requirements on the number of non-faulty and operative processes is a consequence of limiting the adversary to at most a constant fraction of faults, and – similarly as in the main Algorithm 1 – the fact that a random graph of expected degree $\Theta(\log n)$ has dense and shallow (with a small diameter) sub-graphs of size being a constant fraction of all vertices.

Now, the trade-off comes from the fact that the number of random bits needed to solve consensus in a group does not scale linearly with the size of the group. If the size of the group is roughly $\frac{n}{x}$, our optimal algorithm uses $\tilde{O}\left(\frac{n}{x} \cdot \sqrt{\frac{n}{x}}\right) = \tilde{O}\left(\left(\frac{n}{x}\right)^{3/2}\right)$ random bits per run. It follows that performing x of such runs, one by one, gives the desired trade-off between time $T = \tilde{O}\left(x \cdot \sqrt{\frac{n}{x}}\right) = \tilde{O}\left(\sqrt{nx}\right)$ and the randomness complexity $R = \tilde{O}\left(x \cdot \left(\frac{n}{x}\right)^{3/2}\right) = \tilde{O}\left(\frac{n^2}{\sqrt{nx}}\right) = \tilde{O}\left(\frac{n^2}{T}\right)$. The communication bit complexity bound follows from applying the bound of Theorem 5 to each phase of the round-robin scheme separately.

6 Future Directions

We believe that our novel parameterized valency technique could be used in proving lower bounds and impossibility results regarding trade-offs between randomness and other complexity measures in other distributed problems. First important open question is the tight characterization of the tradeoff between round complexity and randomness used by processes for the case when $t = o(n)$. Specifically, Theorem 8 in Appendix D (extended version of Theorem 3) gives an algorithm satisfying the invariant $\texttt{ROUNDS} \times \texttt{RANDOMNESS} = \tilde{\Theta}(n^2)$, regardless of the number of faults. On the other hand, Theorem 2 gives only $\tilde{\Omega}(t^2)$ lower bound on this product.

Second important open question is about trade-off between communication complexity and time when $t = o(n)$, for any type of faults. In particular, for crash failures there is only an upper bound known [23] for such tradeoff, while for omission failures, surprisingly, there is no such tradeoff when $t = \Theta(n)$, by our Theorem 1 (as it simultaneously matches, up to a polylogarithmic factor, the two independent lower bounds on time [10] and communication [1]).

The concept of operative processes, maintaining them locally at (relatively) low cost and using them for performing tasks such as efficient counting and information exchange, could be a game-changing concept in designing and analysis of distributed fault-tolerant algorithms. We demonstrated how to use it efficiently against adaptive omission faults, but we suspect it could be applied to other computation and failure models and problems. On the technical side, it would be interesting to further improve communication performance of using operative processes in case of smaller number of failures.

References

[1] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of Byzantine agreement, revisited. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 317–326. ACM, 2019.

[2] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected o (1) rounds, expected communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[3] Dan Alistarh, James Aspnes, Valerie King, and Jared Saia. Communication-efficient randomized consensus. *Distributed Computing*, 31(6):489–501, 2018.

[4] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2016.

[5] James Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *J. ACM*, 45(3):415–450, 1998.

[6] James Aspnes and Orli Waarts. Randomized consensus in expected O(n $\log^2$ n) operations per processor. *SIAM J. Comput.*, 25(5):1024–1044, 1996.

[7] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), San Diego, California, USA, June 11-13, 2007*, pages 155–164. ACM, 2007.

[8] Hagit Attiya and Faith Ellen. *Impossibility Results for Distributed Computing*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2014.

[9] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley, Second edition, 2004.

[10] Ziv Bar-Joseph and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. In *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 193–199, 1998.

[11] Florent Benaych-Georges, Charles Bordenave, and Antti Knowles. Spectral radii of sparse random matrices. 2020.

[12] Bogdan S. Chlebus, Dariusz R. Kowalski, and Michał Strojnowski. Fast scalable deterministic consensus for crash failures. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 111–120. ACM, 2009.

[13] Benny Chor, Michael Merritt, and David B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *J. ACM*, 36(3):591–614, 1989.

[14] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *J. ACM*, 32(1):191–204, 1985.

[15] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[16] László Erdős, Antti Knowles, Horng-Tzer Yau, and Jun Yin. Spectral statistics of erdős–rényi graphs i: Local semicircle law. 2013.

[17] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[18] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[19] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220, 2003.

[20] Chryssis Georgiou, Seth Gilbert, Rachid Guerraoui, and Dariusz R. Kowalski. Asynchronous gossip. *J. ACM*, 60(2):11:1–11:42, 2013.

[21] Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 96–119. Springer, 2022.

[22] Seth Gilbert and Dariusz R. Kowalski. Distributed agreement with optimal communication complexity. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 965–977. SIAM, 2010.

[23] Mohammad Taghi Hajiaghayi, Dariusz R. Kowalski, and Jan Olkowski. Improved communication complexity of fault-tolerant consensus. In *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 488–501. ACM, 2022.

[24] Anna Karlin and Andrew Yao. Probabilistic lower bounds for byzantine agreement. *Unpublished document*, 1986.

[25] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[26] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable Byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18:1–18:24, 2011.

[27] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[28] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[29] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. 2017.

[30] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[31] Yoram Moses and Sergio Rajsbaum. The unified structure of consensus: a layered analysis approach. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 123–132, 1998.

[32] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[33] Philippe Raïpin Parvédy, Michel Raynal, and Corentin Travers. Strongly terminating early-stopping k-set agreement in synchronous systems with general omission failures. *Theory of Computing Systems*, 47(1):259–287, 2010.

[34] Marcel-Cătălin Roşu. Early-stopping terminating reliable broadcast protocol for general-omission failures. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, page 209, 1996.

[35] Michel Talagrand. Concentration of measure and isoperimetric inequalities in product spaces. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 81:73–205, 1995.

[36] Eli Upfal. Tolerating linear number of faults in networks of bounded degree. In *Proceedings of the eleventh annual ACM symposium on Principles of distributed computing*, pages 83–89, 1992.

Appendix

A Other Related Work

The consensus problem was introduced by Lamport, Pease and Shostak [27, 32], in the context of ***deterministic*** solutions. Fischer, Lynch and Paterson [18] showed that the problem is unsolvable by any deterministic algorithm in an asynchronous setting, even if one process may fail. Fischer and Lynch [17] proved that a synchronous solution requires $t+1$ rounds if up to t processes may crash, which is automatically applicable to settings with more severe failures, including omissions. Dolev and Strong [15] gave an efficient deterministic solutions to Consensus under even a stronger Authenticated Byzantine failures, working in optimal time $t+1$ and using $\Omega(nt)$ messages, which was (nearly) matched by Dolev and Reischuk [14], who proved an $\Omega(t^2+n)$ lower bound.

Recently, Abraham et al. [1] showed that any (even ***randomized***) algorithm that solves Consensus with a constant probability against an adaptive adversary requires $\Omega(t^2+n)$ messages, whp. Regarding time complexity, Bar Joseph and Ben-Or [10] showed that no algorithm can solve Consensus, with correctness probability 1, against an adaptive adversary in fewer than $\Omega(\frac{t}{\sqrt{n \log n}})$ expected rounds. It holds even for more benign crash failures. Other classic work [24] shows a lower bound on the probability of failure of any algorithm that runs in a fixed number of rounds, given that a linear (in n) number of processes is prone to Byzantine failures. This result holds even if the adversary's knowledge is limited to the history of controlled processes and if the algorithm has cryptographic routines such as the public-key infrastructure or threshold signatures. Following a long line of work [1, 19, 21, 25, 29], this result has been recently matched, under these cryptographic assumptions, by an algorithm of [21].

On the algorithms' side, the deterministic algorithm by Dolev and Strong [15] has been also the best known overall solution (including randomized algorithms) against an adaptive adversary (of unbounded computational power and full information) causing omission failures, with respect to time and message complexity. Other works that consider omission failures, sometimes known as general omission failures (since both, incoming and outgoing messages of a faulty processes can be omitted), are mostly concerned with the early-stopping version of the consensus problem variant [33, 34], unlike our work focused on optimization of three complexity measures.

Better time and/or communication complexity is possible if ***failures are more benign,*** e.g., processes are crashed permanently (instead of message omissions) or the ***adversary is weaker*** (more restricted or more oblivious). Hajiaghayi et al. [23] designed a solution under $t < n$ crashes in almost-optimal time and using a subquadratic number of communication bits $\tilde{O}(n^{3/2})$. King and Saia [26] proved that under some limitations on the Byzantine adversary and requiring termination only whp, the subquadratic expected communication complexity $\tilde{O}(n^{3/2})$ and even polylogarithmic

time can be achieved. Even better complexities could be achieved against an *oblivious adversary*, i.e., the adversary who knows the algorithm but has to decide which process to fail and when before the execution starts. Chor, Merritt and Shmoys [13] developed constant-time randomized algorithms while Gilbert and Kowalski [22] solved Consensus in optimal communication complexity $O(n)$ and $\mathcal{O}(\log n)$ time, whp. Sub-quadratic communication is also possible in asynchronous setting under oblivious adversary, which decides processes' and messages' delays in advance, c.f., [20].

On the other side, ***stronger adversaries*** (such as general Byzantine, discussed earlier) or/and ***asynchrony*** require linear time or/and (super-)quadratic communication. Georgiou et al. [20] showed that even a simpler task of fault-tolerant gossip requires linear time *or* quadratic communication when an adaptive adversary crashes processes. Recently, Alistarh et al. [3] showed how to obtain almost-optimal communication complexity $O(n^2 \log n)$ if less then $n/2$ processes may fail, which improved upon the previous result $O(n^2 \log^2 n)$ by Aspnes and Waarts [6] and is asymptotically almost optimal due to the lower bound $\Omega(n^2)$ by Attiya and Censor.

B Almost-optimal Algorithm against Omissions Failures

We present a randomized algorithm that solves consensus with probability 1. It works in $O\left(\frac{t}{\sqrt{n}} \log^2 n\right)$ rounds with high probability and uses $O\left(n(t \log^4 n + n)\right)$ communication bits with probability 1 against any number up to $t < \frac{n}{30}$ of omission-faulty processes. We give a description below while the pseudocode of the algorithm has been provided in Algorithm 1 in Section 3, together with an extended overview and illustrative figures.

The algorithm partitions processes into two classes: *operative* processes and *inoperative* processes. The operative processes have the property that any two of them can exchange a bit of information in at most $O(\log n)$ rounds, using a certain sparse graph of communication. In contrast, the inoperative processes are those that, due to the actions of the adversary, have been excluded from the fast flow of information guaranteed for the operative ones. We note that our partition is dynamic. The status of a process can change during the run of the algorithm. Our partition is also independent of the partition into faulty and non-faulty processes at the logical level. A non-faulty process may become inoperative if its communication neighborhood diminishes (e.g., many of its randomly selected neighbors could become faulty and omit messages), while faulty processes may stay operative as long as they are good transmitters from the perspective of other processes, despite the omissions of some of their messages. The operative/inoperative status of a process is also local – other processes may not be aware of it.

To start, all processes are initially designated as operative (see line 1). Next, each operative process determines its communication neighbors, a set denoted by V_p in the algorithm, based on its identifier p and the set of all processes $\mathcal{P}$ taking part in the protocol. To be precise, in Theorem 4 we state the existence of a sparse random graph with a fault-tolerant properties strong enough to support the later stage of the algorithm regardless of the omissions pattern. Since the processes know $\mathcal{P}$, they can all pre-compute the same graph (i.e., they can all choose a lexicographically smallest such graph) that is guaranteed by Theorem 4. Observe that, in order to do this selection, the processes do not need to communicate, as the properties of such graph are purely combinatorial and only rely on the knowledge of p and $\mathcal{P}$.

During the main loop of the algorithm (lines 5-13), the operative processes work collectively to establish a consensus decision among them. This work is divided into $O\left(\frac{t}{\sqrt{n}} \log n\right)$ *epochs* and utilizes a variant of the majority-vote protocols introduced for the case of, more benign, permanent crash failures in [10].

In our approach, only operative processes participate in the majority-vote protocol. This is achieved as follows: each operative process prepares a variable (interpreted as a *candidate value* for the final decision) b_p initialized to its input bit. In each epoch, the operative processes rely on two different communication subroutines, GroupBitsAggregation and GroupBitsSpreading (see lines 6 and 8), to calculate the number of operative 1's and 0's present in the candidate values. The communication algorithms GroupBitsAggregation and GroupBitsSpreading guarantee that, after $O(\log n)$ rounds and using at most $O\left(n^{3/2}\text{polylog}\,(n)\right)$ communication bits, every operative process has a correct estimation of the number of operative 1's in the candidate values from the beginning of the epoch (the variable $\texttt{ones}_p$ in the pseudocode) and operative 0's in the candidate values from the beginning of the epoch (the variable $\texttt{zeros}_p$ in the pseudocode) after their completion. Since the description of these routines is highly technical, we devoted Subsection B.1 for this purpose. Based on the values of the variables $\texttt{ones}_p$ and $\texttt{zeros}_p$ process p decides what candidate value assign to variable b_p for the next epoch. If the estimation $\texttt{ones}_p$ makes at least $\frac{18}{30}$ fraction of the estimation of all operative processes (which is equal to $\texttt{ones}_p + \texttt{zeros}_p$), then the process assigns $b_p = 1$; if the estimation $\texttt{ones}_p$ is less than half of the estimation of the total number of operative processes then the process assigns $b_p = 0$. In any other case, it assigns b_p to a uniform random bit. The proportion $\frac{18}{30}$ is exactly $\frac{1}{2}$ incremented by the maximal possible fraction of inoperative processes there can be in the algorithm. It guarantees that no two operative processes can deterministically assign $b_p = 0$ and $b_p = 1$ at the same time. See Figure 3 for illustration.

In the analysis, we will demonstrate that if the number of new processes becoming inoperative in a number of consecutive epochs is smaller than $\sqrt{n}$, the choices made by the operative processes will lead to the same candidate value b_p across all operative processes with a constant probability. Our communication algorithms are designed in such a way that $n - \Theta(t)$ processes are always operative in the system, regardless of the adversary's actions. Therefore, after $O\left(\frac{t}{\sqrt{n}}\log n\right)$ such epochs, we can assert that operative processes have reached a consensus decision (i.e., on same value) with high probability. To lift the probability of success to 1, the operative processes adopt a safety rule in the following form. If the estimation value $\texttt{ones}_p$ makes at least $\frac{9}{10}$ fraction of the estimation of all operative processes (or, symmetrically, the estimation of $\texttt{ones}$ makes less than $\frac{1}{10}$ fraction), a process p marks in an auxiliary variable $\texttt{decided}_p$ that it is ready to decide by setting it to *true*. The idea is that, according to the previous arguments, it is only with probability less than $O\left(\frac{1}{n}\right)$ that there is an operative process that has not become ready to decide during the $O\left(\frac{t}{\sqrt{n}}\log n\right)$ epochs. In this unlikely event, processes of this type execute a slow but deterministic consensus protocol given as Theorem 4 in [15], whose $O(t)$ running time is accounted to the total time complexity with $O\left(\frac{1}{n}\right)$ probability only.

To be more precise, in the last part of the algorithm, lines 14-20, the operative processes who have the variable $\texttt{decided}_p$ set to *true*, broadcast their variable b_p (at this time it is a consensus value) to all other processes. Then, the operative processes that have the variable $\texttt{decided}_p$ set to *true* and inoperative processes who have received a consensus value b in the previous round, acquire it, decide on the value and stop from participating in the protocol. On the other hand, all operative processes that have the variable $\texttt{decided}_p$ set to $false$, execute the deterministic consensus protocol from [15], Theorem 4, which reaches consensus with probability 1 in $O(t)$ rounds. In the analysis, we show that if any operative process has the variable $\texttt{decided}_p$ set to *true*, all other operative processes have the same candidate value in their bit b_p as the operative process with the variable $\texttt{decided}_p$ set to *true*, thus – the decision can safely by made on the value of the variable b. The formal analysis of the correctness, running time and bit complexity is given in Subsection B.2.

B.1 Communication patterns of operative processes within an epoch

In this part we describe the communication algorithms used by processes to gather information about candidate values' distribution from the beginning of the epoch. We start by explaining the algorithm GroupBitsAggregation, which achieves the goal limited only to subgroups of processes of size $O(\sqrt{n})$. Limiting the size of a group on which we estimate the candidate values is crucial for achieving $O(n^2)$ communication bit complexity.

Explanation of GroupBitsAggregation. We partition the set of processes $\mathcal{P}$ into at most $\lceil\sqrt{n}\rceil$ groups $W_1, \ldots, W_{\lceil\sqrt{n}\rceil}$, each with a size of at most $\lceil\sqrt{n}\rceil$ processes, as shown in lines 3-4. See also an example in Figure 1. Since every process knows the set of process names, the partition could be predefined, i.e., can be calculated locally by each process using the same local deterministic algorithm. Processes in different groups work independently in algorithm GroupBitsAggregation.

The objective of each group is to compute the number of operative processes within the group that have the variable b set to 0 and the number of those with $b = 1$, or mark inoperative the processes that cannot gather the necessary information. At the end of the algorithm, each operative process p should have these two values, $\texttt{b\_zeros}_p$ and $\texttt{b\_ones}_p$, stored in its local memory, and updated its operative status $\texttt{operative}_p$. Algorithm 2 presents the pseudocode, while a formal description is provided below. For each group W_i, where $1 \leq i \leq \lceil\sqrt{n}\rceil$, we use a binary-tree decomposition of depth $\lceil\log(\sqrt{n})\rceil$. At the lowest layer $L^{(i)}(1)$ of the decomposition, the processes in W_i are divided into $\lceil\sqrt{n}\rceil$ singleton *bags*, yielding the following:

$$L^{(i)}(1) := \{L^{(i)}(1,1), \ldots, L^{(i)}(1, \lceil\sqrt{n}\rceil\}.$$

For any subsequent layer j, $2 \leq j \leq \lceil\log(\sqrt{n})\rceil$, the bags of layer j are formed as a union of bags at level $j-1$ corresponding to the children of the j-level bag in the tree. This way, at the higher levels of the decomposition, the number of bags is halved, but the size of a single bag is doubled. Formally, the bag $L^{(i)}(j,k)$, for $1 \leq k \leq \left\lceil \frac{\sqrt{n}}{2^j} \right\rceil$, which belongs to layer $L^{(i)}(j)$, is the union of bags $L^{(i)}(j-1, 2k-1)$ and $L^{(i)}(j-1, 2k)$. Here, we assume that $L^{(i)}(j-1,k') = \emptyset$ if $k' > \left\lceil \frac{\sqrt{n}}{2^{(j-1)}} \right\rceil$.

The communication scheme is navigated by the structure described above. It proceeds in $\lceil\log(\sqrt{n})\rceil$ stages that correspond to aggregating knowledge within bags that belong to increasingly higher levels of the decomposition.

In the first stage, each operative process p that is a member of a bag $L^{(i)}(1,k)$ at the lowest layer $L^{(i)}(1)$ (for $1 \leq k \leq \lceil\sqrt{n}\rceil$) initializes two variables: $\texttt{b\_ones}_p(1,k)$ and $\texttt{b\_zeros}_p(1,k)$ that keep track of the value of its bit b_p, i.e., $\texttt{b\_ones}_p(1,k) = 1$ if $b_p = 1$, and $\texttt{b\_zeros}_p(1,k) = 1$ if $b_p = 0$; inoperative processes skip this step as their values b are not counted (see lines 1-3). Since at level 1 the bags are singletons, every operative process has the correct number of one and zero candidate values of other operative processes in its bag.

In stage j, for $2 \leq j \leq \lceil\log(\sqrt{n})\rceil$ processes work to gather information of the candidate values of all other operative processes that belong to the same bag in the layer $L^{(i)}(j)$, see the main loop in lines 5-12. For $1 \leq k \leq \left\lceil \frac{\sqrt{n}}{2^j} \right\rceil$, fix a bag $L^{(i)}(j,k)$ of the layer $L^{(i)}(j)$. By the hierarchical structure of the tree, at this point of the execution, the operative processes belonging to the left-child bag of $L^{(i)}(j-1, 2k-1)$ already collected the number of operative 0's and 1's among them. The same is true for processes belonging to the right-child bag $L^{(i)}(j-1, 2k)$. To distribute this information between the operative processes in the bag $L^{(i)}(j,k)$, they use procedure GroupRelay described below.

Specification of the procedure GroupRelay. The procedure takes five inputs: the identifier

of a process p that executes it, its operative status $\texttt{operative}_p$, its group W_i, the bag $L^{(i)}(j,k)$ in which p currently exchanges information, and the set $\texttt{counts}_p$, which is the information to be distributed among other operative processes in $L^{(i)}(j,k)$. The information could be initialized differently by the algorithm calling the procedure GroupRelay, depending on whether p belongs to a left-child or a right-child of the bag $L^{(i)}(j,k)$ in the lower layer j, as shown in lines 7-8 of Algorithm 2.

During the execution of the procedure GroupRelay, the processes in W_i have two different roles: *non-faulty* processes of the entire group serve as *transmitters*, while *operative* processes in the bag $L^{(i)}(j,k)$ have an additional role of a *source*. The following 3-round protocol is executed. In the first round, each source sends its set `count` to all transmitters. The transmitters collect received sets `count` and prepare new sets of the same name `count` that are the union of the former. There are at most four logically different values that processes (sources) of the bag $L^{(i)}(j,k)$ try to disseminate, which are the counts of operative zeros and ones originating in the two bags that are children of the bag $L^{(i)}(j,k)$ in the lower level of the decomposition. In case there are two or more different counts of operative zeros and ones for a children bag, the transmitters may choose arbitrary of them. Therefore, each newly prepared set has the size of at most $O(\log n)$ bits. In the second round, the transmitters confirm to sources if they received a message in the previous round. Any source that receives less than $\frac{1}{2}|W_i|+1$ confirmations becomes inoperative. In the third round, the transmitters send the new sets `count` to the sources. Again, the sources that receive less than $\frac{1}{2}|W_i|+1$ notifications in this round become inoperative. If a process becomes inoperative, it stops serving as a source, although it is still considered a potential transmitter in subsequent calls of the algorithm. The change in the operative status of a process is reported to the main algorithm OptimalOmissionsConsenus via variable $\texttt{operative}_p$. Example of how the above 3-round relay process works is given in Figure 2.

After the procedure GroupRelay terminates, each process adds the received counts of operative ones and zeros and proceeds to the next stage of GroupBitsAggregation. Therefore, after the last stage, the operative processes gather the knowledge about the number of operative processes having $b=0$ and the number of those who have $b=1$ in the entire group W_i (as this set corresponds to the bag being the root of the tree). These numbers are stored in variables $\texttt{b\_zeros}_p(\lceil\sqrt{\log n}\rceil, 1)$ and $\texttt{b\_ones}_p(\lceil\sqrt{\log n}\rceil, 1)$, and together with the operative status of a process, they are returned to the main algorithm OptimalOmissionsConsenus.

Explanation of GroupBitsSpreading. The algorithm GroupBitsSpreading serves as a tool to disseminate the values calculated by the grouped operative processes (in the run of the algorithm GroupBitsAggregation) to all operative processes. The processes locally prepare an array of pairs `BitPacks` of size $\lceil\sqrt{n}\rceil$ as follows. If W_ℓ is the group of a process in the partitioning $W_1,\ldots,W_{\lceil\sqrt{n}\rceil}$, then the process writes the numbers of operative processes of W_ℓ having $b=0$ and having $b=1$, which is the result of the algorithm GroupBitsAggregation, at the position ℓ of the array, keeping all other entries empty. Next, the communication is performed along links corresponding to the predetermined graph G with certain combinatorial properties, which existence is guaranteed in Theorem 4, cf., line 2 of Algorithm 1. Recall that G is such that every process $p\in\mathcal{P}$ has degree $O(\log n)$. The set V_p of neighbors is passed as a parameter to every execution of the algorithm GroupBitsSpreading. See also Figure 1 for an illustration of the group partitioning with additional graph spanned on nodes.

The communication in the GroupBitsSpreading algorithm lasts $8\log n$ rounds. Initially, p sends to every process in V_p the entries of the array $\texttt{BitPacks}_p$ that have not been sent via the link yet. When receiving a message from another process $q\in V_p$ process p updates its array $\texttt{BitPacks}_p$ by replacing empty entries with the non-empty entries of the array $\texttt{BitPacks}_q$. Additionally, p

Algorithm 2: GroupBitsAggregation

input: $W_i, p, \texttt{operative}_p; b_p$

1 **if** $operative_p = true$ **then**
2 **if** $b_p = 0$ **then** $\texttt{b\_ones}_p(1,i) \leftarrow 0, \texttt{b\_zeros}_p(1,i) \leftarrow 1$;
3 **else** $\texttt{b\_ones}_p(1,i) \leftarrow 1, \texttt{b\_zeros}_p(1,i) \leftarrow 0$;
4 **end**
5 **for** *stage* $j \in \{2, \ldots, \lceil \log n \rceil\}$ **do**
6 let $L^{(i)}(j,k)$ be p's bag in j-th layer $L^{(i)}(j)$;
7 **if** $p \in L^{(i)}(j-1, 2k-1)$ **then** $\texttt{counts}_p \leftarrow \{\texttt{b\_ones}_p(j-1,2k-1), \texttt{b\_zeros}_p(j-1,2k-1)\}$;
8 **else** $\texttt{counts}_p \leftarrow \{\texttt{b\_ones}_p(j-1,2k), \texttt{b\_zeros}_p(j-1,2k)\}$;
9 $\texttt{counts}_p, \texttt{operative}_p \leftarrow$ GroupRelay$(W_i, p, \texttt{operative}_p, L^{(i)}(j,k), counts_p)$;
10 compute values of variables $\texttt{b\_ones}_p(j-1,2k-1), \texttt{b\_zeros}_p(j-1,2k-1), \texttt{b\_ones}_p(j-1,2k)$, $\texttt{b\_zeros}_p(j-1,2k)$ based on the elements in the set $\texttt{counts}_p$;
11 $\texttt{b\_ones}_p(j,k) \leftarrow \texttt{b\_ones}_p(j-1,2k-1) + \texttt{b\_ones}_p(j-1,2k)$;
12 $\texttt{b\_zeros}_p(j,k) \leftarrow \texttt{b\_zeros}_p(j-1,2k-1) + \texttt{b\_zeros}_p(j-1,2k)$
13 **end**
14 **return** $\texttt{b\_ones}_p(\lceil \log n \rceil, 1), \texttt{b\_zeros}_p(\lceil \log n \rceil, 1), \texttt{operative}_p$

keeps track of the processes from V_p who failed to deliver a message in the most recent round and excludes them from the communication and refutes to accept messages from them in any future round of the algorithm GroupBitsSpreading. If the number of processes from which p received a message is less than $\Delta/3$, where Δ is the parameter from Theorem 4, p becomes inoperative. It stays idle to the end of the current epoch and in all future epochs of the execution of the main algorithm OptimalOmissionsConsensus.

After $8 \log n$ rounds, each operative process adds up the values in its array BitPacks corresponding to the numbers of 0's and 1's, resp., in different groups, and returns these two numbers. The key property, which we prove formally in the analysis of the protocol, is that any two values returned by the operative processes (either the count of 0's or 1's) may differ by at most the number of processes that have turned inoperative during the epoch.

Algorithm 3: GroupBitsSpreading

input: $V_p, p, \ell, \texttt{operative}_p; \texttt{g\_ones}_p, \texttt{g\_zeros}_p$

1 $\texttt{BitPacks}_p \leftarrow [(\emptyset,\emptyset), \ldots, (\emptyset,\emptyset)]$, an array of pairs (initially of zeros) of size $\lceil \sqrt{n} \rceil$;
2 $\texttt{BitPacks}_p[\ell] \leftarrow (\texttt{g\_ones}_p, \texttt{g\_zeros}_p)$;
3 **for** $8 \log n$ *rounds* **do**
4 **for** $q \in V_p$:
  send a message to q containing the entries of $\texttt{BitPacks}_p$ not shared with q before, provided q has not been disregarded earlier
  receive a message (if any) with $\texttt{BitPacks}_q$ sent by q:
  if q has not sent a message, disregard sending to q in any future round;
  for $i \in [\lceil \sqrt{n} \rceil]$: $\texttt{BitPacks}_p[i] \leftarrow \texttt{BitPacks}_p[i] \vee \texttt{BitPacks}_q[i]$;
5 **if** *number of received messages is less than* $\Delta/3$ **then**
  $\texttt{operative}_p \leftarrow false$;
  stay idle until line 8;
6 **end**
7 $\texttt{ones}_p, \texttt{zeros}_p \leftarrow$ the sum of the first (or the second resp.) elements of each non-empty pair of $\texttt{BitPacks}_p$;
8 **return** $\texttt{ones}_p, \texttt{zeros}_p, \texttt{operative}_p$

B.2 Analysis of algorithm OptimalOmissionsConsensus

The building blocks of an execution of the algorithm are *epochs* – full iterations of the main loop of Algorithm 1, OptimalOmissionsConsensus. As any final decision of a process comes from a communication with an operative process, see lines (15, 19), our goal is to prove that, with probability 1, the operative processes store the same value in variables b_p. To achieve that, we will gradually analyze how consecutive subroutines executed within an epoch influence the values b_p held by operative processes.

Analysis of GroupBitsAggregation **algorithm.** In this algorithm, the operative processes work in groups that are given by the partition $\{W_1, \ldots, W_{\lceil\sqrt{n}\rceil}\}$ of the set $\mathcal{P}$. Consider a group W_i. For a process p and a process q such that p, q in W_i, we say that q *contributes* to variable $\texttt{b\_ones}_p(j,k)$ ($\texttt{b\_zeros}_p(j,k)$ respectively), for some $1 \leq j \leq \lceil\log(\sqrt{n})\rceil$, $k \geq 1$, if the input b_q is counted in the variable $\texttt{b\_ones}_p(j,k)$ (or ($\texttt{b\_zeros}_p(j,k)$ depending on the value of b_q).

Lemma 1. *Let $M_i \subseteq W_i$ be the set of operative processes in the group W_i at the end of the algorithm* GroupBitsAggregation. *Any process $p \in M_i$ contributes to both variables, $\texttt{b\_ones}_q(\lceil\log(\sqrt{n})\rceil, 1)$ and $\texttt{b\_zeros}_q(\lceil\log(\sqrt{n})\rceil, 1)$, of any other process $q \in M_i$.*

Proof. In the proof we inductively show that any two processes p', q' belonging to the intersection of a bag $L^{(i)}(j,k)$ and the set M_i have the property that p' contributes to variables $\texttt{b\_ones}_{q'}(j,k)$, $\texttt{b\_zeros}_{q'}(j,k)$ of the process q', where $1 \leq j \leq \lceil\log(\sqrt{n})\rceil$, $k \geq 1$. The induction proceeds with respect to the depth of a layer, j, in the tree decomposition $L^{(i)}$ of the group W_i. For the base case, we observe that the lemma holds for any pair of processes belonging to the set $M_i \cap L^{(i)}(1,k)$, where $L^{(i)}(1,k)$ is a bag in the first layer, simply because $M_i \cap L^{(i)}(1,k)$ is a singleton set (in this case, the only feasible pair has $p' = q'$).

For the inductive step, consider a bag $L^{(i)}(j,k)$ in some layer $j > 1$. Let p' and q' be any two processes belonging to $M_i \cap L^{(i)}(j,k)$. During the execution of the GroupRelay procedure, the process p', who had the role of a source, relayed its set `count` to at least $\frac{1}{2}|W_i| + 1$ other processes (transmitters), because otherwise it would have lost its operative status. Similarly, q' has received the accumulated set `count` from at least another $\frac{1}{2}|W_i| + 1$ processes. However, these two sets of transmitters must have at least one common transmitter, who in this case assures that q' received the counts of p'. Since $L^{(i)}(\lceil\log(\sqrt{n})\rceil, 1) = W_i$, the lemma is proven. □

Next, we bound the bit complexity of the algorithm GroupBitsSpreading when limited to a single group only.

Lemma 2. *Processes in a single group use at most $O(n \log^2 n)$ bits of communication, in total, in an execution of the algorithm* GroupBitsAggregation.

Proof. The algorithm takes $O(\log n)$ stages of communication. In a stage, the procedure GroupRelay is executed to distribute information between processes belonging to every bag of the layer of the stage. The information about a single bag distributed in the messages of the procedure GroupRelay is of size $O(\log n)$ bits at most, as it contains at most four counts of size $O(\log n)$ each, c.f. the description of the procedure. The messages send in the procedure are always between transmitters and sources. This first class of processes contains at most $O(\sqrt{n})$ processes (the size of any group). The second amortizes to $O(\sqrt{n})$ if considered the union of all bags on the layer corresponding to the stage. It follows then, that at most $O(n)$ messages is sent per stage, in total. Thus the lemma follows. □

Analysis of GroupBitsSpreading **algorithm.** We first explain formally the scheme of communication used in the algorithm GroupBitsSpreading. Let $\mathcal{R}(n,\rho)$ be a random graph on n vertices in which every edge is contained independently with probability ρ. For suitable parameters α and ℓ, the following properties of certain graphs $\mathcal{R}(n,\rho)$ will be used to analyze omission-tolerance of the predetermined graph G used for communication.

Definition 1 (*of expansion and edge-sparsity*)**.**

- ***Expansion:*** *A graph G is said to be* ℓ-expanding*, or to be an* ℓ-expander*, if any two subsets of ℓ nodes are connected by an edge.*
- ***Edge-sparsity:*** *A graph G is said to be* (ℓ,α)-edge-sparse *if for any set $X \subseteq V$ of* at most *ℓ nodes, there are at most $\alpha|X|$ edges internal for X.*

To assure that there exists a graph G that can be used by processes to perform the communication in the algorithms we provide the following theorem.

Theorem 4. *Let $n \in \boldsymbol{N}$, and $\Delta := 832 \log n$. A random graph $\mathcal{R}(n, \Delta/(n-1))$ satisfies all the below properties whp:*
(i) *it is $(n/10)$-expanding,*
(ii) *it is $(n/10, \Delta/15)$-edge-sparse,*
(iii) *the degree of each node is between $\frac{19}{20}\Delta$ and $\frac{21}{20}\Delta$.*

Proof. In this proof we will use Chernoff bounds in the following form, where $\varepsilon > 0$ and $X = \sum_{1 \le i \le k} X_i$ is a sum of independent Bernoulli trials with the expected value μ:

$$\Pr[X \ge (1+\varepsilon)\mu] < \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right)^{\mu} \tag{1}$$

$$\Pr[X \ge (1+\varepsilon)\mu] < e^{-\mu\varepsilon^2/3}, \text{ for } \varepsilon \le 1 \tag{2}$$

$$\Pr[X \le (1-\varepsilon)\mu] < e^{-\mu\varepsilon^2/2}, \text{ for } \varepsilon < 1 \tag{3}$$

see [30].

We first prove that property (iii) is satisfied by graph H whp. The expected node degree in H is Δ. By the Chernoff bounds (2) and (3), each degree of H is between $(19/20)\Delta$ and $(21/20)\Delta$ with probability at least

$$1 - 2 \cdot \exp\left\{-\Delta \cdot (1/20)^2/3\right\} \ge 1 - 1/n$$

as $\Delta > 832 \log n$.

Now we consider property (ii). Consider a set $X_1 \subseteq V$ of at most $n/10$ nodes. We show that it has at most $|X_1|\,\Delta/15$ internal edges with probability at least $1-1/n^2$. Let x_1 stand for $|X_1|$. Recall that the expected number of edges internal for X_1 is $\Delta/(n-1) \cdot x_1\,(x_1-1)\,/2$. The probability that the number N of internal edges is more than $x_1\Delta/15$ can be upper bounded by the Chernoff bound (2) with $\varepsilon_1 = 1/3$ to obtain

$$\Pr\left[N > (1+\varepsilon_1) \cdot \frac{\Delta x_1\,(x_1-1)}{2(n-1)}\right] \le e^{-\frac{\Delta x_1 (x_1-1)}{2(n-1)} \cdot (\varepsilon_1)^2/3} \le e^{-\Delta x_1/270},$$

as $x_1 \le n/10$. It follows that the probability that a set X_1 of at most x_1 nodes with more than $x_1\Delta/15$ internal edges exists in graph H, where $x_1 \le n/10$, is at most

$$\sum_{x_1=1}^{n/10} \binom{n}{x_1} e^{-\Delta x_1/270} \le \sum_{x_1=1}^{n/10} 2^{x_1 \log(n) - \Delta x_1/189} \le \sum_{x_1=1}^{n/10} 2^{-3x_1 \log n} \le 1/n^2,$$

as $\Delta > 832 \log n$.

It remains to prove property (i). Consider any two disjoint sets X and Y of exactly ℓ elements each. The expected number of edges connecting two disjoint sets of nodes X and Y is $\Delta|X||Y|/(n-1)$. Therefore, by the Chernoff bound (3), we obtain that the number N of edges connecting the sets X and Y is less than $2/3\Delta|X||Y|/(n-1)$ with a probability that is at most

$$\Pr[N < (1-1/3) \cdot \Delta|X||Y|/(n-1)] \leq e^{-\Delta|X||Y|/(n-1)\cdot(1/3)^2/2} \leq e^{-\Delta n/1800}$$

Consider an event that two sets of nodes of $n/10$ elements each and with less than $2/3\Delta|X||Y|/(n-1)$ edges between them exist. The probability of this event is at most

$$\binom{n}{\frac{n}{10}} \binom{n}{\frac{n}{10}} e^{-\Delta n/1800} \leq 2^{2n/10 \cdot \lg n} \cdot e^{-\Delta n/1800} \leq 2^{n/5 \log n} \cdot 2^{-832\Delta n/1800} \leq 2^{-2n \log(n)/15} < 1/n^2.$$

Since the complement of this event guarantees that every two disjoint set of size $n/10$ have at least $2/3\Delta|X||Y|/(n-1) > 1$ connecting edges, thus $(n/10)$-expansion follows. □

Let us now fix a particular graph G that satisfies the above properties and let this graph be the same as the one on which the processes decide when executing line 2. As a consequence of these properties, it can be also shown that close neighborhoods of any vertex in G contain regular subgraphs and grow rapidly. Let us start by giving the appropriate definition.

Definition 2 (*of* (γ, δ)*-dense-neighborhood*)**.** *For a node $v \in V$, denote $N_G^d(v)$ the set of vertices at distance at most d from v in graph G. Then, for $\gamma \geq 1$, a set $S \subseteq N_G^\gamma(v), v \in S$ 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.*

To formalize the meaning of the rapid growth of a close neighborhood of a vertex in G, we provide the following lemma.

Lemma 3. *If graph $G = (V, E)$ of n nodes is $(n/5, \Delta/15)$-edge-sparse then any $(\gamma, \Delta/3)$-dense-neighborhood for a node $v \in V$ has at least $\min\{2^\gamma, n/10\}$ nodes, for $\gamma \geq 0$.*

Proof. Consider a node $v \in V$ and a set S which is a $(\gamma, \Delta/3)$-dense-neighborhood for v. Let A_i stand for $S \cap N_G^i(v)$. We show the following fact by induction on i, where $1 \leq i \leq \gamma - 1$: the set A_i has at least $\min\{2^i, n/10\}$ elements. The base of induction for $i = 1$ follows directly from the property that $v \in S$ has at least $\frac{\Delta}{3} > 3$ neighbors in $A_1 = S \cap N_G^1(v)$.

The inductive step: suppose that the inequality $|A_i| = |S \cap N_G^i(v)| \geq \min\{2^i, n/10\}$ holds for i, where $1 \leq i < \gamma - 1$. We prove the inequality for $i + 1$ as follows. If A_i has at least $n/5$ elements, then we are done. Otherwise the set A_i has at most $|A_i|\Delta/15$ internal edges, by the $(n/10, \Delta/15)$-edge-sparsity. We obtain that there are at least $|A_i|\Delta/3 - 2 \cdot |A_i|\Delta/15 = 3|A_i|\Delta/15$ edges between the sets A_i and $N_G^{i+1}(v) \setminus A_i$. Let X denote the set of neighbors of A_i in $N_G^{i+1}(v)$ A_i. It follows that the number of edges internal for $A_{i+1} = A_i \cup X$ is at least $3|A_i|\Delta/15$. On the other hand, this number is at most $(|A_i| + |X|)\Delta/15$, by $(n/10, \Delta/15)$-edge-sparsity. Consequently, the inequality $(|A_i| + |X|)\Delta/15 \geq 3|A_i|\Delta/15$ holds. Hence $|X| \geq 2|A_i|$ and there are at least $2^{i+1} \geq \min\{2^{i+1}, n/10\}$ elements in A_{i+1} by the inductive assumption. We use the fact that $A_{\gamma-1} = S \cap N_G^{\gamma-1}(v)$ has at least $\min\{2^{\gamma-1}, n/10\}$ elements. As the inequality $2^{\gamma-1} \geq n/5$ can be verified directly, the proof is complete. □

We note here that statements of similar results to Theorem 4 and Lemma 3 can be found in [12], however, the theorem is given without a proof, and the lemma is proven for different constants. Therefore, for the sake of completeness, we provided the proofs of the versions needed in this work.

An important property[11] is that removal of a linear fraction of nodes from the graph G cannot leave many nodes with small neighborhood, as is stated next:

Lemma 4. *If graph $G = (V, E)$ satisfies the properties of Theorem 4, then for any set of nodes $T \subset G$ of size at most $\frac{n}{15}$ there exists a subset A of G of size at least $n - \frac{4}{3}|T|$ such that:*
(i) $A \cap T = \emptyset$, (ii) *every node from A has at least $\frac{\Delta}{3}$ neighbors in A.*

Proof. Fix any set of nodes $T_1 \subset V$ of size at most $\frac{n}{10}$ and consider the following, inductive, definition. Given a set T_i, for $i \geq 1$, we define $T_{i+1} = T_i + \{v\}$ if there exists a node $v \in V \setminus T_i$ that has at least $\frac{37}{60}\Delta$ neighbors in T_i; otherwise define $T_{i+1} = T_i$. By the fact that there is a finite number of nodes in G and by the fact that if $T_{j+1} = T_j$ then all subsequent sets $T_{j+2}, T_{j+3}, \ldots$ are the same, there must be exactly one index K such that for all $k \geq K$ it holds $T_{k+1} = T_k$, but $T_K \neq T_{K-1}$.

We prove that $K < |T_1|/3$. Assume to the contrary that $K \geq |T_1|/3$. Let $k = |T_1|/3$. Based on this assumption, observe that adding a new node v to T_i, for $1 \leq i \leq K-1$, increases the number of edges induced by the set T_i by $37\Delta/60$ at least. It follows that the number of edges induced by T_K is at least $37K\Delta/60$. On the other hand, the number of vertices of T_K is $|T_1| + k - 1 \leq 4/3|T_1| = 4/3 \cdot n/15 = 4n/45$. Therefore, we can use the $(n/10, \Delta/15)$-edge-sparsity property of G to the set of vertices T_k and conclude that there can be at most $4/3|T_1| \cdot \Delta/15 = 4\Delta K/45$ edges in the subgraph induced by T_K. Comparing the last upper bound of $4\Delta K/45$ on the number of edges with the derived lower bound bound of $37\Delta K/60$ yields a contradiction as $37\Delta K/60$ is not less than $4\Delta K/45$. It follows that K must be smaller than $|T_1|/3$.

To finish the proof of the lemma, we define A as $V \setminus T_K$. We first observe that degrees of all nodes in A are at least $\Delta/3$. If there was a node of a smaller degree, it would have to have at least $\frac{19}{20}\Delta - \frac{1}{3}\Delta \geq \frac{37}{60}\Delta$ neighbors in T_K and thus it would be added to T_{K+1} violating the definition of K. Since $K < |T_1|/3$, thus $|T_K| \leq 4|T_1|/2$ as there is only one node added at the time. The size of A must be then at least $n - 4|T_1|/3$ which completes the proof. □

Establishing the necessary properties of the communication graph used in runs of the GROUP-BITSSPREADING algorithm, we focus on analyzing how information is spread across the graph in these runs. Let us fix a run of the algorithm. For this run, we define $\texttt{OP}_i$ to be the set of these processes who maintained their status as operative until the end of the round i of the run. Observe that there is $8 \log n$ rounds, corresponding precisely to the iterations of the main loop of the algorithm, thus $\texttt{OP}_{8\log n}$ denotes the set of processes that finished the run as operative. We define $G_i^{\texttt{OP}}$ a subgraph of G with the set of vertices equal to $\texttt{OP}_i$ and the set of edges equal to the links by which processes in $\texttt{OP}_i$ received messages in round i of the run of this algorithm. We call a link *operative* in round i if it belongs to $G_i^{\texttt{OP}}$. Observe the a link operative in round i correctly transmits messages in both directions in rounds $1, \ldots, i-1$. That is because links that are observed faulty by a process are never used in the future rounds.

Lemma 5. *For a fixed run of the algorithm `GroupBitsSpreading`, any processes p in $\texttt{OP}_{8\log n}$ has a $(2\log n, \Delta/3)$-dense-neighborhood in the graph $G^{\texttt{OP}}_{8\log n}$.*

Proof. For proving the existence of a $(2\log n, \Delta/3)$-dense-neighborhoods for process p, we use the following inductive construction. Let $R_0(p)$ be the set of these processes from which p receives

[11] A similar property has been previously observed for graphs of bounded spectral radius, c.f. [36]. There are known results that bounds spectral radius of a random Erdos-Renyi graph, c.f. [11, 16], and in consequence could lead to the same conclusion. However, for the sake of the completeness, we provide an alternative, fundamental proof that relies solely on combinatorial properties guaranteed by Theorem 4.

a message in the last round of the algorithm algorithm GroupBitsSpreading. Observe that $|R_0(p)|$ is at least $\Delta/3$ as p is in $\mathtt{OP}_{8\log n}$. Let B_1 be a subgraph of G with the set of vertices equal to $B_0 \cup R_0(p)$ and the set of edges equal to the links by which p receives a message from $R_0(p)$ in the last round of the algorithm. First, we note that the graph B_1 is a subgraph of $G^{\mathtt{OP}}_{8\log n-1}$. Any processes of B_1 must be operative until round $8\log n-1$ as only operative processes send messages. Also, operative processes use links that continuously deliver messages, c.f. 4, instruction 3, hence any link that delivers a message *to* process p in round $8\log n$ has to deliver a message in rounds $1,\ldots,8\log n-1$ *from* p. It also holds that, as observed, $|B_1| \geq \Delta/3$, and thus the graph B_1 satisfies the definition of $(1,\Delta/3)$-dense-neighborhood of p in the graph $G^{\mathtt{OP}}_{8\log n-1}$. This concludes the base case of the inductive construction.

For the inductive step, we define the set $R_i(p)$, for $2 \leq i \leq 2\log n$, as a set of these processes that send a message to any process in B_{i-1} in the round $8\log n - i$ of the algorithm. Let B_i be a subgraph of G with the set of vertices equal $R_i(p)$ and the set of edges equal to the set of link by which a process from B_i received a message in the round $8\log n - i$. The same argument as the one used in the base case shows that B_i is a subgraph of $G^{\mathtt{OP}}_{8\log n-i}$. By the inductive assumption $B_{i-1} \subseteq G^{\mathtt{OP}}_{8\log n-i+1}$. Also, from the line 4 it follows that the operative subgraphs $G^{\mathtt{OP}}$ are downward monotonic and thus $G^{\mathtt{OP}}_{8\log n-i+1} \subseteq G^{\mathtt{OP}}_{8\log n-i}$. Therefore $B_{i-1} \subseteq B_i$. Also, since $B_{i-1} \subseteq G^{\mathtt{OP}}_{8\log n-i+1}$, thus processes from B_i are operative until the round $8\log n - i + 1 \geq 6\log n$. It follows that each of these processes receives at least $\Delta/3$ messages in the round $6\log n$ and these messages must be send from processes in B_i. Therefore B_i satisfies the definition of $(i,\Delta/3)$-dense-neighborhood of p in $G^{\mathtt{OP}}_{8\log n-i}$. This completes the induction step and concludes the proof. □

We say that a process $p \in \mathtt{OP}_{8\log n}$ can *reach* a process $q \in \mathtt{OP}_{8\log n}$ during the run of the algorithm GroupBitsSpreading, if there exists a path of processes $p = s_1, s_2, \ldots, s_k = q$, such that the process s_{i+1} received a message from s_i in ith round of the run of the GroupBitsSpreading algorithm, for any $1 \leq i \leq k-1$.

Lemma 6. *For a run of the algorithm* `GroupBitsSpreading`*, any processes p in $\mathtt{OP}_{8\log n}$ can reach any other process q in $\mathtt{OP}_{8\log n}$.*

Proof. Since p and q belong to $\mathtt{OP}_{8\log n}$, thus by Lemma 5 there exist $(2\log n,\Delta/3)$-dense-neighborhoods of, respectively, p and q in the graph $G^{\mathtt{OP}}_{6\log n}$. Denote the neighborhoods S_p and S_q respectively. Since $G^{\mathtt{OP}_{6\log n}}$ is a subgraph G, thus S_1 and S_2 are also subgraphs of G. This observation allows us to use Lemma 3 and conclude that the sizes of S_1 and S_2 are $n/10$ at least. Since there is at most $n/30$ faulty processes, the are subsets S_1^C, S_2^C of S_1 and S_2 consisting of only non-faulty processes of size at least $n/10$ each. By the $(n/10)$-expanding property of G there is at least one edge connecting S_1^C and S_2^C. Let p' and q' be the endpoint processes of this edge. Now, we can define the path by which p can reach q. Processes p and p' belong to $G^{\mathtt{OP}}_{6\log n}$. By the previous observation links of this subgraph transmit messages bidirectionally in rounds $1,\ldots,2\log n$. Since p' is in a $(2\log n,\Delta/3)$-dense-neighborhood of p in $G^{\mathtt{OP}}_{6\log n}$, thus p' is in distance at most $2\log n$ from p in $G^{\mathtt{OP}}_{6\log n}$. It follows that p can reach p' in $2\log n$ rounds. In the round $2\log n+1$ the process p' can reach q' as they are both non-faulty and operative and there is an edge between them. Then, in the next $2\log n$ rounds, q' can reach q by the same reasoning as for p and p'. Thus, the lemma is proven. □

Analysis of the main algorithm. In the final part of the proof, we connect together the properties of the algorithms GroupBitsAggregation and GroupBitsSpreading and explain

how they lead to correct updates of values b in lines 9- 11 of the main algorithm OptimalOmissionsConsensus, resulting eventually in a valid consensus decision. We start by noting that the communication graphs used in any of the inner algorithms are dense enough to maintain a large fraction of correct processes operative, regardless of the actions of the adversary. We recall here the assumption that the number t of faulty processes is less than $\frac{n}{30}$. Also, recall that we call a single iteration of the main loop of the algorithm OptimalOmissionsConsensus an epoch. For an epoch, let `OP_END` denote the set of these processes that are operative at the end of this epoch.

Lemma 7. *For any epoch, the size of the set `OP_END` is larger than $n - 3t$.*

Proof. Consider an epoch $\mathcal{E}$ of the algorithm and let F be the set of processes that the adversary corrupted in the epoch $\mathcal{E}$ or before. Recall the partition $\{W_1, \ldots, W_{\lceil\sqrt{n}\rceil}\}$ of the set $\mathcal{P}$ used in the algorithm GroupBitsAggregation. Since $|F| \leq t$, processes then by a counting argument there exists a set of $\sqrt{n} - 2\frac{t}{\sqrt{n}}$ groups, which corresponds to a set of at least $n - 2t$ processes, such that every process in the set is non-faulty and has more than half non-faulty processes in its group. Denote this set X.

Consider the subgraph induced by X in the graph G used for the communication in GroupBitsSpreading algorithm. Using Lemma 4 for the set $G \setminus X$, which has size at most $2t \leq n/15$, we conclude that there exists $X' \subseteq X$ that has size at least $n - 4/3 \cdot 2t = n - 8/3t \geq n - 3t$ such that every process from X' has degree at least $\Delta/3$ in X'.

We finish the proof by arguing, that all process from X' belongs to the set `OP_END` for the epoch $\mathcal{E}$. First, every process p in X' also belongs X thus, by the definition of X, it always receives more than half of messages from its group in all the executions of GroupBitsAggregation that occur before or in the epoch $\mathcal{E}$. Second, we choose X' to consist only of non-faulty processes. Communication on the links *between* them is always reliable. By the fact that every processes of X' has degree at least $\Delta/3$, we conclude that every processes from X' maintains its operative status in all the executions GroupBitsSpreading that occur before or in the epoch $\mathcal{E}$. Therefore, $X' \subseteq$ `OP_END`. □

We say that a process p *contributes* to the sum of ones (or zeros) of a process q if its bit b_p is included in the value $ones_q$ calculated in line 8 of the main algorithm OptimalOmissionsConsensus ($zeros_q$ resp.).

Lemma 8. *For any epoch, every process p contained in the set `OP_END` with $b_p = 1$ contributes to the sum of ones of any other process q from `OP_END`. Analogically, if $b_p = 0$ then p contributes to the sum of zeros of any other process q from `OP_END`.*

Proof. Consider any $p \in$ `OP_END`. Let W_i be the p's group, i.e. $p \in W_i$. By Lemma 1, we get that any operative process r contributes to the values returned by any other operative process within its group in the algorithm GroupBitsAggregation. This yields, that whatever pair of values $(\texttt{g\_ones}_i, \texttt{g\_zeros}_i)$, describing the number of 0's and 1's among operative process in W_i, the process q receives during the execution GroupBitsSpreading algorithm, the bit value b_p contributes to this pair. On the other hand, Lemma 6 assures that p can reach q. It follows that q receives at least one pair of values $(\texttt{g\_ones}_i, \texttt{g\_zeros}_i)$, and the lemma is proven. □

Lemma 9 (Lemma 4.3 in [10])**.** *Assume that n processes independently choose a random bit from uniform distribution. Let X be the random variable denoting the number processes that chose bit*

1. *Then for any* $t \leq \sqrt{n}/8$

$$\Pr(X - \mathbb{E}(X) \geq t\sqrt{n}) \geq \frac{e^{-4(t+1)^2}}{\sqrt{2\pi}} .$$

We call an epoch *good* if at most $\sqrt{n}$ processes become inoperative during the epoch. The following explains how lines 9- 11 change b values of operative processes under the assumption of a sequence of consecutive good epochs.

Lemma 10. *Consider three consecutive good epochs* $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$. *Let* $\texttt{OP\_END}_i$, *for* $i \in \{1, 2, 3\}$ *be the set of operative processes at the end of the epoch* $\mathcal{E}_i$. *With probability* $\Omega(1)$ *all processes belonging to* $\texttt{OP\_END}_3$ *store the same value* b *before the third epoch ends.*

Proof. Using Lemma 7, we have that $|\texttt{OP\_END}_1| \geq n - 3t \geq \frac{9}{10}n$, since we assumed $t < \frac{1}{30}n$. Combining Lemma 8 with the assumption that in the epoch $\mathcal{E}_1$ at most $\sqrt{n}$ change their status to inoperative, we see that values $ones_p$, $zeros_p$ and $ones_p + zeros_p$ calculated in a process p in epoch $\mathcal{E}_1$ differ by at most $\sqrt{n}$ from the corresponding values of any other processes from $\texttt{OP\_END}_1$. Therefore, in the epoch $\mathcal{E}_1$ no two processes can execute line 9 and line 10 at the same time, since the difference between right-hand-sides of these two inequalities is at least $\frac{1}{30}|\texttt{OP\_END}_1| > \sqrt{n}$.

We first consider the case when no process executes line 10, that is processes either assign a random bit to b or 0. Observe that if a processes assigns 0 with probability 1 instead of $\frac{1}{2}$ it is more likely that the total number of zeros exceeds a certain threshold. Therefore by applying Lemma 9[12] we get that with a constant probability C_1, the number of processes starting the next epoch $\mathcal{E}_2$ with b equal zero is at least $\frac{|\texttt{OP\_END}_1|}{2} + 3\sqrt{n}$. The constant C_1 is an appropriate constant in Lemma 9 corresponding to the fact that we have a $|\texttt{OP\_END}_1| \geq \frac{9}{10}n$ lower bound on the size of $\texttt{OP\_END}_1$ and we measure the deviation of size $\sqrt{n}$ from the mean. Nevertheless, it follows that with probabiblity C_1 the number of processes having $b = 1$ is at most

$$|\texttt{OP\_END}_1| - \frac{|\texttt{OP\_END}_1|}{2} - 3\sqrt{t} \leq \frac{|\texttt{OP\_END}_1|}{2} - 3\sqrt{n} .$$

In the next epoch $\mathcal{E}_2$ at most $\sqrt{n}$ new processes become inoperative and thus the ratio of the number of 1's held by the operative processes to the total number of operative processes must remain lower than $\frac{1}{2}$ in the entire epoch $\mathcal{E}_2$. It follows then, that all operative processes execute line 10 and assign b to 0 before the epoch ends.

The second case is when at least one process executes line 9 in the epoch $\mathcal{E}_1$. If there is at least one operative process assigning 1 to its variable b without sampling, and at most $\sqrt{n}$ processes become inoperative the epoch $\mathcal{E}_1$, thus every other operative process assigns 1 to the variable b or it assigns a random value. Let x be the number of processes that assign a random value in the epoch $\mathcal{E}_1$. We proceed with two subcases:

- **Subcase 1:** Assume that $x \geq \frac{8}{10}|\texttt{OP\_END}_1|$. By Lemma 9, with a constant probability C_2, more than $\frac{4}{10}|\texttt{OP\_END}_1| + 3\sqrt{n}$ from processes who execute line 11 assign 0 to their value b. Thus, by the beginning of the epoch $\mathcal{E}_2$ at most

$$|\texttt{OP\_END}_1| - \frac{4}{10}|\texttt{OP\_END}_1| + 3\sqrt{n} = \frac{6}{10}|\texttt{OP\_END}_1| - 3\sqrt{n}$$

[12] Although Lemma 9 concerns the number of 1's being chosen by processes, if the distribution is uniform then 0 and 1 have the same probability of occurring and, by the symmetry, the same lemma can be used to estimate the number of chosen 0's.

processes have value b set to 1. In the epoch $\mathcal{E}_2$ at most $\sqrt{n}$ processes become inoperative, therefore the ratio of *ones* to the total number of operative counted in any process will be smaller than $\frac{6}{10}$. Thus, we arrived at the case when no process can execute line 9 in the epoch $\mathcal{E}_2$ and the two consecutive epochs are good. Using the same argument as for the first case of this lemma, we get that with a constant probability all operative processes have value b set to 0 before the epoch $\mathcal{E}_3$ ends.

- **Subcase 2:** Suppose that $x < \frac{8}{10}|\texttt{OP\_END}_1|$. By Lemma 9 there is a constant probability that at most $\frac{4}{10}|\texttt{OP\_END}_1| - 3\sqrt{n}$ processes set the value of b to 0. This is because at most $\frac{8}{10}|\texttt{OP\_END}_1|$ processes assign a random value and all the remaining ones assign 1 to the variable b. Therefore, at least

$$|\texttt{OP\_END}_1| - \frac{4}{10}|\texttt{OP\_END}_1| + 3\sqrt{n} = \frac{6}{10}|\texttt{OP\_END}_1| + 3\sqrt{n}$$

processes start the epoch $\mathcal{E}_2$ with b set to 1. Again, in the epoch $\mathcal{E}_2$ at most $\sqrt{n}$ processes can become inoperative. In such a case, Lemma 8 proves that operative processes at least $\frac{6}{10}|\texttt{OP\_END}_1| + 2\sqrt{n}$ counts of the value 1 and thus all operative processes must execute line 9 in the epoch $\mathcal{E}_2$. Therefore all operative processes assign value 1 to their variable b.

□

In the following lemma we show how the operative processes assure termination on the same value for non-faulty processes, even in the case of the unlikely event that some of the operative processes do not set the variable `decided` to *true*.

Lemma 11. *With probability* 1*, all non-faulty processes decide and the decision is on the same value.*

Proof. Let O be the set of the processes that are operative after the last epoch of the algorithm OptimalOmissionsConsensus. We can further divide the set O into two disjoint classes. The class $D \subseteq O$ of processes that have the variable `decided` set to *true* and the class $U \subseteq O$ consisting of these processes that have the variable `decided` set to $false$.

First, we prove that regardless of the actually partition of the operative processes into the classes D and U all non-faulty processes decide. Lemma 7 assures that $|O| \geq n - 3t \geq \frac{9n}{10} \geq 2t$, thus either D or U have size greater than t.

If it is the case that $|D| > t$, then there must be at least one non-faulty processes in D. The existence of this process assures that every non-faulty processes from the set $\mathcal{P} \setminus O$ receives a message in the line 15 of the algorithm, and in consequence decides. The remaining non-faulty processes are all operative and they decide in line 16 in the case they belong to the set D, or in line 18 in the case they belong to U. The termination in the latter case follows from the correctness of the deterministic protocol from [15], Theorem 4. For the case $|U| > t$ we have the following reasoning. The non-faulty operative processes can either belong to D or U. Every non-faulty process belonging to D decides in line 16. All non-faulty processes belonging to U cannot decide in lines 14-16 and they will execute line 18. From the termination of the deterministic protocol used in line 18, the non-faulty processes from U will reach a decision and propagate the decision to all remaining processes that do not terminated yet. Since $|U| > t$, thus in the propagation takes part at least one non-faulty process which guarantees that every non-faulty processes will receive a decision in line 19. This gives that eventually all non-faulty processes decide.

Second, we show that all decision are on the same value. To this end, we argue that if there exists a process p that has the variable $\texttt{decided}_p$ set to *true* when the last epoch ends then all operative processes have the value of the variable b the same as the process p. Consider the epoch in which p, by executing line 12, sets the variable $\texttt{decision}_p$ to *true*. It follows that the counts $\texttt{ones}_p$ and $\texttt{zeros}_p$ satisfy $\texttt{ones}_p > \frac{27}{30}(\texttt{ones}_p + \texttt{zeros}_p)$ (or $\texttt{ones}_p < \frac{3}{30}(\texttt{ones}_p + \texttt{zeros}_p)$, but since both cases are symmetric, we analyze only the first one). The number of operative process is always at least $n - 3t \geq \frac{9}{10}n$, by Lemma 7. By Lemma 8, it follows that the values of variables $\texttt{ones}$ and $\texttt{zeros}$ stored by different operative processes in the same epoch can differ by at most $4t$. Therefore, the ratio $\texttt{ones}/(\texttt{ones} + \texttt{zeros})$ calculated in any other operative process in this epoch is at least

$$\frac{\texttt{ones}_p - 4t}{\texttt{ones}_p + \texttt{zeros}_p + 4t}.$$

Since the process p executes the line 12, we get that the lower bound $\texttt{ones}_p \geq \frac{9}{10}(\texttt{ones}_p + \texttt{zeros}_p)$. It follows that

$$\frac{\texttt{ones}_p - 4t}{\texttt{ones}_p + \texttt{zeros}_p + 4t} \geq \frac{\frac{9}{10}\left(\texttt{ones}_p + \texttt{zeros}_p\right) - 4t}{\texttt{ones}_p + \texttt{zeros}_p + 4t} \geq \frac{18}{30},$$

where the last inequality follows from a simple calculation, taking into account that $\texttt{ones}_p + \texttt{zeros}_p \geq \frac{9}{10}n$ and $t \leq \frac{1}{30}n$. It yields, that the inequality in line 12 holds for any other operative process in this epochs. Subsequently, this gives that any operative process sets the variable b to 1, by executing line 9, if p sets the variable $\texttt{decide}_p$ to *true* in this epoch. By examining lines 14-20 we see that any process can decide only on a value that originates in a operative process. Since the consensus protocol used in line 18 satisfies the validity condition, we have that its decision, if any, must also be equal to the decision of any operative process. This way there is only one value propagated as the decision in the system and every non-faulty process must decide on this value regardless of when it receives the value.

It remains to a situation in which all operative processes have the variable $\texttt{decided}$ set to $false$. In this case, no process can decide before reaching line 18. The decision in any process is the outcome of the consensus protocol employed in line 18 and by correctness of the employed algorithm, it must be the same across all process that decide. □

Finally, we give the main theorem.

Theorem 5. *The algorithm* OptimalOmissionsConsensus *solves with probability* 1 *consensus against the adaptive adversary capable of controlling* $t < \frac{n}{30}$ *processes. With high probability the number of rounds used by the algorithm is* $O\left(\frac{t}{\sqrt{n}}\log^2 n\right)$ *rounds and the number of communication bits is* $O\left(n\left(t\log^3 n + n\right)\right)$. *With probability* 1 *it uses* $O\left(t\sqrt{n}\log^2 n\right)$ *random bits.*

Proof. Let us first argue about the correctness of the algorithm. The termination and agreement conditions the consequence of Lemma 11. The validity condition follows from the following observation. If all processes start an epoch with the same value of the variable b, either 0 or 1, then with probability 1 all operative processes will have the same value of the variable b (consistent with the initial bit) in all subsequent epochs. This is because in the algorithms GroupBitsAggregation and GroupBitsSpreading processes count only this values that where present in a process at the beginning of the epoch. Consequently, in such a case no process ever accesses a random source in line 11 of the algorithm OptimalOmissionsConsensus, as the ratio of 1's of operative processes to all values is always either 0 or 1. Since the decision of any non-faulty algorithm is always derived from a value of the variable b of an operative process, c.f. lines 14-20, thus the validity follows.

We now give the analysis of the number of rounds, the number of bits and the number of random bits used by the algorithm. By Lemma 7, there are at most $4t$ processes that might become inoperative during the run of the algorithm. On the other hand, any epoch that is not good has more than $\sqrt{n}$ processes that become inoperative in this epoch. Since the algorithm executes $\frac{t}{\sqrt{n}} \log n$ epochs, thus by the pigeonhole principle there is at least $\Omega(\log n)$ disjoint sequences of three consecutive *good* epochs.

By Lemma 10, after each such triple, there is a $\Omega(1)$ probability that all operative processes have the same value of the variable b. Since, as argued above in this proof, if the operative processes start an epoch with the same value of the variable b they start with same value of the variable b in the following epoch, thus the probability that after all such good sequences of epochs, the operative processes do not have the same value in variable b is at most $O(1)^{\log n} = \frac{1}{n^C}$, for some absolute constant $C > 0$. Equivalently, it holds that with probability $1 - \frac{1}{n^C}$ the operative processes have the same value in the variable b in the end of the last epoch. Moreover, with this probability, all operative processes have also the variable `decision` set to *true*. This holds because if a triple of good epochs ends strictly before the last epoch, then all values of the variable b stored by the operative processes are the same, and in the next epoch all operative processes must set the variable `decided` to *true* when executing line 12.

Assuming that the operative processes store *true* in their variable `decided`, then all operative processes execute line 14 of the algorithm. Since there is at least $n - 3t$ operative processes and at most $t < \frac{1}{30}n$ faulty processes, in the next round each non-faulty process receives the value of at least one operative process and, in result, adopts the value as its decision. It follows that in line 16 all non-faulty processes decide. The operative processes because they all have the variable `decided` set to *true*; the non-faulty processes that are not operative decide because they received a message from an operative one. Since each epoch takes $O(\log n)$ rounds and there is $\frac{t}{\sqrt{n}} \cdot \log n$ epochs, we have that all non-faulty processes terminate in $O\left(\frac{t}{\sqrt{n}} \log^2 n\right)$ rounds whp.

To upper bound the communication bit complexity, we first analyze how many bits are sent per epoch. The algorithm GroupBitsAggregation uses $O(n \log^2 n)$ bits per group, as it is given in Lemma 2. Since the algorithm is executed in parallel on $\lceil \sqrt{n} \rceil$ groups, this results in $O(n^{3/2} \log^2 n)$ bits per call to this algorithm by the main one. The algorithm GroupBitsSpreading takes $O(\log n)$ rounds of communication on a graph with the maximum degree $O(\log n)$, according to Theorem 4. Per each link of the communication graph, processes send at most $O(\sqrt{n} \log n)$ bits, amortized – in a round, each process sends only the information that has not been passed yet and the total amount of information corresponding to the size of the array `BitPacks`, which contains $O(\sqrt{n} \log n)$ bits. Since there are $O(\frac{t}{\sqrt{n}} \log n)$ epochs, the communication bits' complexity incurred by processes during all epochs is $O(t \cdot n \log^3 n)$. The consensus protocol from [15] (Theorem 4.) incurs at most $O(n^2 t)$ additional bits in its communication, but this can happen with probability at most $\frac{1}{n^C}$ for some absolute constant C. On the other hand, the number of bits used for informing the inoperative processes is $O(n^2)$ with probability 1, as each operative process broadcasts its value at most once. Thus, the bound on the communication bit complexity follows.

For the upper bound on number of random bits, we observe that the use of those is predetermined by the number of epochs. Each operative process uses at most one random bit in an epoch, thus the total number of random bits used by all processes is bounded by $O\left(t\sqrt{n} \log^2 n\right)$. □

B.3 Comparison to the state-of-the art strategies for solving consensus in the model of crash failures

When the power of the adversary is limited to crashing processes, meaning that a process becomes completely disconnected from other processes at some point of the execution (which is a decision of the adversary), there exist optimal, or almost-optimal, algorithms with respect to time and bit complexity.

As for the time efficient strategies, most of them rely on the time-optimal algorithm proposed by Bar-Joseph and Ben-Or [10], who employed the idea of the random coin in the case where there is no clear majority of either preferred value: 0 or 1, among the processes. The use of the randomness in the case of crashes shares the same intuition with the approach of omissions. Because the standard bounds on the deviation from the mean on many i.i.d. random variables guarantee that, with constant probability, either the number of 0's exceeds the numbers of 1's by $\Theta(\sqrt{n})$ or vice-versa. In any case, the adversary is expected to crash $\Omega(\sqrt{n})$ processes to maintain decision uncertainty preserve the status quo in the system. Nevertheless, the important difference in the case of crash failures is that a crashed process stops to communicate with other processes starting from the round when the crash has occurred or the next round – hence, every correct process can assume the same default value of the failed process with at most one round delay. In consequence, the adversary cannot prolong the strategy of crashes mitigating the deviations from the mean for more than $O(\sqrt{n})$ rounds as it would require controlling to many processes. This crucial property does not hold when omissions are allowed. A process, controlled by the adversary, may deliberately avoid communication with some chosen process (which would force the chosen process to assume a default input value of the faulty one), while at the same time – informing some set of other processes about its actual value, which may be different from the default one. Even more maliciously, a faulty process may change the set of processes it communicates with from round to round which allows the adversary for even more flexibility. This motivates the partition of processes into classes that are downgrade monotonic rather than of unpredictable behavior. A contribution of our paper is to design such a partition, into operative / inoperative processes, and show that the random coin idea can be efficiently implemented on this partition.

The state-of-the-art bit efficient strategy was proposed by Hajiaghayi et al. [23]. They designed a randomized consensus algorithm against crash failures, which uses $O\left(n^{3/2} \cdot \text{polylog}\,(n)\right)$ bits whp and terminates in the almost-optimal time $O\left(t/\sqrt{n} \cdot \text{polylog}\,(n)\right)$ (but has no provably almost-optimal communication bit complexity), by exploiting certain "locally compact" properties of expander graphs of gradually growing degree, used for scheduling communication between processes. Unfortunately, the approach in [23] is not efficient against the omission failures controlled by the adaptive, full-power adversary. Similarly to [10], [23] also relies on the fact that processes permanently stop after crashes, which allows them to amortize time or communication to the number of fail-stops, e.g., by doubling the number of contacted processes each time when too few responses are received. This no longer works against omissions, because the adversary can control incoming/outgoing messages of the process that implements such doubling strategy, and enforce that the process inquires $\Theta(n)$ other processes before the adversary allows it to receive any messages. This way even a single omission-faulty process may contribute linearly to the communication complexity, while for crash failures such contribution could be amortized to a polylogarithm per crashed process.

C A Lower Bound

We give a new lower bound, announced in Theorem 2, showing that any consensus algorithm achieving T round complexity with high probability, has to make at least $\Omega\left(\frac{t^2}{\log n}\right)$ calls to a random source, with probability at least $1 - \frac{1}{\log n}$. The main probabilistic tool used in the analysis of the lower bound strategies, introduced later in this section, is an abstract one-round coin-flipping game.

The coin-flipping game, revisited. The sides of the coin-flipping game are: k players, an adversary and a function f that decides the outcome of the game and is known to both the adversary and players. The game has the following organization. First, the players propose input values. The value of a player p is drawn from an arbitrary distribution X_p, independently from other players. Next, the adversary looks at the drawn values and has the power to *hide* some subset of them (which we will also refer to as "failing players" or "taking over players"). The hidden values are denoted by $\perp$. Finally, the outcome of the game is a binary value determined by the evaluation of the function $f : \{X_1 \cup \perp\} \times \ldots \times \{X_k \cup \perp\} \to \{0, 1\}$ in the point that corresponds to actions of the players and the adversary. We say that a sequence of players' values $y = (y_1, \ldots, y_k) \in X_1 \times \ldots \times X_k$ can be *biased towards a value* $v \in \{0, 1\}$ if the adversary can change some players' values to $\perp$ obtaining a sequence $y' \in \{X_1 \cup \perp\} \times \ldots \times \{X_k \cup \perp\}$ such that $f(y') = v$.

Application of the coin-flipping game to the lower bound proof. The usefulness of this game for proving our lower bound is best visible if the processes are seen as players, the values X_p are possible state transitions of k (out of n) processes that do random calls, the "hiding" action of the adversary is to omit all links of the process in the round corresponding to the game, and the binary outcome of the function is a predetermined, by the adversary, classification of executions (i.e., executions that are more likely to output 1 vs those that are more likely to output 0).

The coin-flipping game – technical part. In [10], the authors show that with probability at least $1 - \frac{1}{n}$, an arbitrary game of n players can be biased towards a particular outcome if the adversary can hide / fail $\Omega(\sqrt{n \log n})$ players. Below, by applying parameterized (by probability α) Talagrand's concentration inequality, we generalize this result and show that even if only $k = o(n)$ players use randomness, the probability of biasing the game can be exponentially (in k) close to 1. In contrast, in [10] this relation is linear and flat in the number of all players. Formally, in our proof we will rely on the Alon's and Spencer's formulation of Talagrand's inequality for convex bodies that originally appeared in [35], adjusted to the notation used in our paper.

Theorem 6 (Theorem 7.6.1 in [4])**.** *Let Ω be a probability space and let*

$$\Omega^k = \Omega \times \Omega \times \cdots \times \Omega$$

be a product probability space. Then, for any $U \subseteq \Omega^k$ and for any $t \geq 0$, it holds that

$$\Pr[U] \cdot \Pr\left[U_t^c\right] \leq e^{-t^2/4} \ ,$$

where U_t^c is the complement of U_t defined as follows:

$$U_t = \{x \in \Omega^k \ : \ \rho(U, x) \leq t\}$$

and where $\rho(A, x)$ is the Talagrand's convex distance defined as

$$\rho(U, x) = \max_{h, \|h\|_2 \leq 1} \rho_h(U, x) \ , \quad \text{where} \quad \rho_h(U, x) = \min_{y \in U} \sum_{i \ : \ x_i \neq y_i} h_i$$

for $h = (h_1, \ldots, h_k) \in \mathbb{R}^k$ and $x, y \in \Omega^k$.

Lemma 12. *For any $\alpha \leq \frac{1}{2}$, one can bias the single-round coin-flipping game towards one particular outcome $v \in \{0,1\}$, with probability greater than $1-\alpha$, by hiding at most $8\sqrt{k\log(\alpha^{-1})}$ players' values.*

Proof. First observe that if $k = 0$ then there is only one outcome of the game which happens with probability 1. Therefore, the lemma is trivially proven. In the remainder, we assume that $k \geq 1$.

Consider the set $U^0 \subseteq X_1 \times \ldots \times X_k$ which corresponds to those sequences that *cannot* be biased towards the outcome 0 by replacing at most $8\sqrt{k\log(\alpha^{-1})}$ values with the default value $\perp$. If $\Pr(U^0) \leq \alpha$ then the lemma is proven with $v = 0$.

Assume that $\Pr(U^0) > \alpha$. Let $B(U^0, t)$ be the set of the sequences from $X_1 \times \ldots \times X_n$ that differ with an element of U^0 on at most t values. We apply Talagrand's concentration inequality (in the version of Theorem 6) to prove that $\Pr\left(B\left(U^0, 8\sqrt{k\log(\alpha^{-1})}\right)\right) \geq 1-\alpha^{15}$. Observe that for vector h such that $h_i = \frac{1}{\sqrt{k}}$, we get that $\|h\| = 1$ and thus

$$B\left(U^0, 8\sqrt{k\log(\alpha^{-1})}\right) = \{x : \rho_h(U^0, x) \leq 8\sqrt{\log(\alpha^{-1})}\} \supseteq \{x : \rho(U^0, x) \leq 8\sqrt{\log(\alpha^{-1})}\}\ , \quad (4)$$

where the first equality follows from the fact $h_i = \frac{1}{\sqrt{k}}$, in which case $\rho_h(U, x)$ is just a weighted Hamming distance, and the latter inclusion from the fact that for any x it holds $\rho_h(U^0, x) \leq \rho(U^0, x)$. From Theorem 6 we get that

$$\Pr\left(U^0\right) \cdot \Pr\left(\{x : \rho(U^0, x) \leq 8\sqrt{\log(\alpha^{-1})}\}^c\right) \leq \exp\left(-(8\sqrt{\log(\alpha^{-1})})^2/4\right) = \alpha^{16}\ .$$

Since we assumed $\Pr\left(U^0\right) > \alpha$, then we can rewrite

$$\Pr\left(\{x : \rho(U^0, x) \leq 8\sqrt{\log(\alpha^{-1})}\}^c\right) \leq \alpha^{16} \cdot \Pr\left(U^0\right)^{-1} \leq \alpha^{15}\ ,$$

which together with Property (4) implies

$$\Pr\left(B\left(U^0, 8\sqrt{k\log(\alpha^{-1})}\right)\right) \geq \Pr\left(\{x : \rho(U^0, x) \leq 8\sqrt{\log(\alpha^{-1})}\}^c\right) \geq 1-\alpha^{15}\ .$$

To finalize the proof of the lemma, we observe that if a sequence y belongs to $B\left(U^0, 8\sqrt{k\log(\alpha^{-1})}\right)$, it differs with an element $b \in U^0$ on at most $8\sqrt{k\log(\alpha^{-1})}$ positions. Let b' be derived from b by replacing those positions with $\perp$. Since $b \in U^0$, therefore $f(b') = 1$ which in turn gives that $f(y') = 1$ where y' is derived from y by replacing the same positions with $\perp$. Therefore, $y \notin U^1$ as it can be biased towards 1 by changing at most $8\sqrt{k\log(\alpha^{-1})}$ positions. We therefore have that $\Pr(U^1) \leq 1 - \Pr\left(B\left(U^0, 8\sqrt{k\log(\alpha^{-1})}\right)\right) \leq \alpha^{15}$, where the last inequality follows from the above use of Talagrand's inequality. Since $\alpha^{15} < \alpha$ for $\alpha < \frac{1}{2}$, thus the lemma is proven. □

Consider now a set of n processes and how their internal changes of states affect the state of the entire execution. We use Lemma 12, with $\alpha = n^{-3}$, to limit the actions of an adversary when only a subset of processes choose to evoke randomness when changing their states between two communication rounds. In the context of an execution of a consensus algorithm, we have the following.

Corollary 1. *Consider a single-round coin-flipping game on a set of n processes from which only $0 \leq k \leq n$ rely on random choices when changing their internal state while all others use deterministic transitions. Then, by failing at most $8\sqrt{k\log^3 n}$ processes, an adversary can bias the outcome of the game towards 0 or 1 [13] with probability at least $1 - \frac{1}{n^3}$.*

[13] Note that the values 0 and 1 here are not necessarily tied to a consensus decision, but rather mark two different states of the whole system of processes.

Description of adversarial strategies. We say that a consensus algorithm is *p-strongly-correct*, for $p \in (0,1]$, if it satisfies all three conditions together (i.e., agreement, validity and termination) with probability at least p against any adaptive adversary.[14] In the following, we restrict ourselves only to $1-\frac{1}{n^{3/2}}$-strongly-correct algorithms. Recall also that each algorithm is structured into rounds that are further split into two phases: a local computation phase (in which calls to a random source are included) and a communication phase. Without loss of generality, we can assume that a local computation phase begins with making a decision if a process is ready to decide. We keep track of an execution of a given algorithm only until the first process decides. Consider an execution $\mathcal{E}$ of an algorithm. For any round i of the execution, we introduce the following notation – let

- $\mathcal{H}_i$ be the algorithm *history* of the whole execution, taken at the beginning of the local computation phase of round i (aka the *state* of the algorithm),
- r_i be the number of processes that, based on their local subset of $\mathcal{H}_i$, decide to use random bits in the local computation phase of the current round,
- $\mathcal{A}_i$ be an adversarial strategy in round i and the subsequent rounds, under the history $\mathcal{H}_i$ (we will drop sub-index i if it is clear from the context),
- $\Pr(\mathcal{H}_i, \mathcal{A})$ be the probability of reaching consensus on value 1 when continuing the run of the algorithm with history $\mathcal{H}_i$ under adversarial strategy $\mathcal{A}$.

We restrict the adversary to strategies in which it fails at most $16\sqrt{r_i \log n} + 1 = \Theta(\sqrt{r_i \log n} + 1)$ processes in a round i. Next, we introduce the classification of states of an execution, based on their potential valency – the concept introduced in [18] for deterministic executions and in [10] for randomized ones. The main difference in our classification, compared to [10], is that we use the stronger Corollary 1 in the analysis and therefore we could refine the definition of bi-valent states to exclude scenarios that have too low probability of finishing either with decision value 1 or 0. This refinement, see below, is necessary to be able to analyze the amortized number of accesses to the random source.

Types of states (based on valency). We say that a state $\mathcal{H}_i$ is:

- *null-valent* if for all adversarial strategies $\mathcal{A}$ we have $\frac{1}{n \log n} - \frac{i}{n^2} \leq \Pr(\mathcal{H}_i, \mathcal{A}) \leq 1 - \frac{1}{n \log n} + \frac{i}{n^2}$,
- 1-*valent* if there is an adversarial strategy $\mathcal{A}$ such that $\Pr(\mathcal{H}_i, \mathcal{A}) > 1 - \frac{1}{n \log n} + \frac{i}{n^2}$ and for every other adversarial strategy $\mathcal{A}'$: $\Pr(\mathcal{H}_i, \mathcal{A}') \geq \frac{1}{n \log n} - \frac{i}{n^2}$,
- 0-*valent* if there is an adversarial strategy $\mathcal{A}$ such that $\Pr(\mathcal{H}_i, \mathcal{A}) < \frac{1}{n \log n} - \frac{i}{n^2}$ and for every other adversarial strategy $\mathcal{A}'$: $\Pr(\mathcal{H}_i, \mathcal{A}') \leq 1 - \frac{1}{n \log n} + \frac{i}{n^2}$,
- *bivalent* if there are adversarial strategies $\mathcal{A}, \mathcal{A}'$ such that $\Pr(\mathcal{H}_i, \mathcal{A}) > 1 - \frac{1}{n \log n} + \frac{i}{n^2}$ and $\Pr(\mathcal{H}_i, \mathcal{A}') < \frac{1}{n \log n} - \frac{i}{n^2}$.

An execution that is 1-valent or 0-valent is also called *uni-valent*. We remark that the classes are disjoint and cover the whole space of an algorithm's states.

[14] The adjective "strongly" is to distinguish from a weaker version in which each property must hold *separately* with probability at least p.

Overview of the lower bound proof. Initially, we show that there is an initial assignment of input bits to processes such that the algorithm starts in the state either bivalent or null-valent (c.f. Lemma 13). Consequently, we will prove that if the algorithm starts a round i in an null-valent or bivalent state, then there is an adversarial strategy $\mathcal{A}$ that keeps the algorithm state in one of these two classes for one more round, with high probability, or in the round every such strategy has reached the limit of t processes failed by the adversary (Lemmata 14, 15). In the latter case, we show that because the strategy in each round failed at most $8\sqrt{r_i \log n}$ processes when $r_i \geq 1$ (or 1 process if $r_i = 0$), then a standard application of the Cauchy-Schwartz inequality yields that the desired amount of random calls has been used by the algorithm, with a constant probability. We now continue with the detail proof of Theorem 2.

Lemma 13. *For any synchronous consensus algorithm there exists an initial state, which, if the adversary can control one process, is null-valent or bivalent.*

Proof. Assume to the contrary, that every initial state (i.e., every assignment of input values to processes) is either 1-valent or 0-valent. Certainly, the state a_1 where all inputs are 0 has to be 0-valent, while the state b_1 with only 1's as an input has to be 0-valent. This holds, because the algorithm starting from all 0's has to solve consensus correctly (satisfying validity condition) with output 0 with probability $1 - \frac{1}{n^{3/2}}$ at least (and the opposite is true for all 1's). Consider now a chain of states $a_1 = s_1, s_2, \ldots, s_n = b_1$ such that each two consecutive states are different only on one input value. Observe, that there must be a state s_j in the sequence that is 0-valent and the next state s_{j+1} is 1-valent. Let p be the process that gets different input in these two states. Consider the executions starting with s_j but if the adversary fails the process p and does not send its messages. If the execution becomes null-valent or bi-valent then the lemma is proven. If the execution remains 0-valent, then the executions s_{j+1} is bi-valent. It is 1-valent by definition, however, by controlling (i.e. failing) the process p and stopping it from sending messages it becomes 0-valent, since there is no difference between this execution and the one started from s_j with the adversary failing the same process. The same argument can be conducted if failing the process p in the execution s_j leads to 1-valent execution. Since both these facts hold with high probability, their intersection also does and yields a contradiction. Thus, the lemma is proven. □

Lemma 14. *A state $\mathcal{H}_i$ that is null-valent at the beginning of round $i < n$ can be extended to a null-valent state at the end of the round with probability greater than $1 - \frac{2}{n^2}$ by failing at most $16\sqrt{r_i \log n}$ processes.*

Proof. Consider a one-round coin-flipping game applied to the state $\mathcal{H}_i$ under the randomness used by the set of r_i processes. Let us partition the possible outcomes (that is possible states in the subsequent round) of the game into two exhaustive classes:

Class (a) the states that are either bivalent or 1-valent;

Class (b) the states that are either null-valent or 0-valent.

By Corollary 1, the adversary can bias the game by failing at most $8\sqrt{r_i \log n}$ processes towards either class (a) or (b), with probability at least $1 - \frac{1}{n^2}$. If the adversary could bias towards class (a), obtaining a state H_{i+1} in the subsequent round , then by the classification, there exists a further strategy $\mathcal{A}_{i+1}$ that guarantees

$$\mathbb{P}\mathrm{r}(\mathcal{H}_{i+1}, \mathcal{A}_{i+1}) > 1 - \frac{1}{n \log n} + \frac{i+1}{n^2} \ .$$

It follows that at the beginning of round i there was a strategy $\mathcal{A}'_i$ that guaranteed

$$\begin{aligned}\mathbb{P}\mathrm{r}(H_i, \mathcal{A}'_i) &> \left(1 - \frac{1}{n^3}\right)\left(1 - \frac{1}{n\log n} + \frac{i+1}{n^2}\right) \\ &= 1 - \frac{1}{n\log n} + \frac{i+1}{n^2} - \frac{1}{n^3} + \frac{1}{n^4\log n} - \frac{i+1}{n^5} \\ &\geq 1 - \frac{1}{n\log n} + \frac{i}{n^2}\ ,\end{aligned}$$

where for the last inequality we used $i < n$, which contradicts the fact that $\mathcal{H}_i$ is null-valent. Therefore, by failing at most $8\sqrt{r_i \log n}$ processes before the communication phase of round i, the adversary can steer to a *temporary* state H'_i satisfying (b). Here, "temporary" refers to the fact that the state is measured between two phases of the round i. Conditioned on this, we can view the remaining random choices of still-non-faulty processes as another one-round coin-flipping game with at most r_i processes using randomness. We can further classify the outcomes (states) of this game into two classes:

Class (a'): the states that are 0-valent;

Class (b'): the states that are null-valent.

The adversary, by failing at most $8\sqrt{r_i \log n}$ other processes (thus altogether at most $16\sqrt{r_i \log n}$ processes), still between the phases of round i, can bias towards one of those new classes with probability at least $1 - \frac{1}{n^3}$, again by Corollary 1. By analogous reasoning as above, we can exclude the class (a'), as biasing towards this class would mean that the state $\mathcal{H}_i$ was 0-valent. Therefore, it must be that the execution can be biased toward class (b') and henceforth the state $\mathcal{H}_i$ can be extended to a null-valent state $\mathcal{H}_{i+1}$ in the next round with sufficiently high probability. □

Next, we analyze the strategy for a state that is bivalent. In the proof, we use the fact that for any value of r_i, the upper bound $16\sqrt{r_i \log n} + 1$ on the number of processes that the adversary can fail in a round is at least 1.[15]

Lemma 15. *Let $\mathcal{H}_i$ be a bivalent state. By failing at most $16\sqrt{r_i \log n} + 1$ processes per round, the adversary can extend the state for the next $i' > 1$ rounds, with probability at least $1 - \frac{i'}{n\log n}$, reaching a state $\mathcal{H}_{i+i'}$ that is either bivalent or terminating. The latter case can happen only because failing the necessary processes in round $i + i' - 1$ would exceed the limit t on the total number of failures.*

Proof. Consider the state $\mathcal{H}_i$ and let processes complete their local computation phase scheduled for this round. Let $\mathcal{H}'_i$ be the random variable equal to the new state after the local computation phase. Since the communication phase involves no additional randomness, we can distinguish the following cases:

Case A. The randomness used by processes (in round i) led to the next state $\mathcal{H}'_i$ that is either null-valent or bivalent — then the adversary does nothing.

Case B. The result of the random choices (in round i) of the processes led to an uni-valent state $\mathcal{H}'_i$, say 1-valent.

[15] Note that the first part of the upper bound formula, $16\sqrt{r_i \log n}$, which was sufficient for extending null-valent state as in Lemma 14, could be 0 for $r_i = 0$, therefore we need $+1$ in the upper bound formula on the number of processes that could be failed by an adversary in a round, in order to give the adversary the power to fail at least one process in each round.

Nevertheless, since the state $\mathcal{H}_i$ is bivalent, there exists a strategy $\mathcal{A}_i$ that achieves $\mathbb{P}\mathrm{r}(\mathcal{H}_i, \mathcal{A}_i) < \frac{1}{n \log n} - \frac{i}{n^2} < \frac{1}{n \log n}$ (where the probability includes all possibilities for the value of the random variable $\mathcal{H}'_i$). Next, we shall prove that the fact that $\mathcal{H}'_{i'}$ is 1-valent, together with the existence of the strategy $\mathcal{A}_i$, disallows termination of the algorithm with probability greater than $1 - \frac{1}{n}$ in the round $i+1$, and in some cases – even in later rounds, unless the adversary reached the limit of failed processes.

If 0 is to be decided in the next round, then the state must become either 0-valent or bivalent at some point of implementing $\mathcal{A}_i$. If it becomes bivalent, then the adversary stops implementing $\mathcal{A}_i$. At this point, abandoning the strategy and letting the algorithm continue, leads to a bivalent state $\mathcal{H}_{i+1}$ which proves the lemma.

The other case is when during implementing $\mathcal{A}_i$, the state becomes 0-valent. Assume that this happens after the failure of a process after all its messages are delivered. Then, the adversary does not fail this process and stops implementing the strategy. It follows that crashing or not crashing the process switches between 0-valent and 1-valent states. Therefore, the initial state of the round $i+1$ must be bivalent, as the adversary can fail the process at the very beginning of round $i+1$ and switch between the two types. (Here we use a fact that the adversary has to be able to fail at least 1 process per round, which is why we set up the upper bound $16\sqrt{r_i \log n} + 1$ that is at least 1 even for $r_i = 0$.) It can be also the case that the state becomes 0-valent after failing a process together with failing to deliver some non-empty subset of its messages. Let u be the recipient of the message that – if delivered – would keep the state 1-valent, while if failed – it would change the state to 0-valent. The adversary stops the algorithm from delivering the message but does not take any further actions in round i. Now, failing the process u at the beginning of round $i+1$ hides the information whether the last message has been delivered or not, which again was the only difference between 0-valent and 1-valent state – thus the state must be bivalent, proving the lemma.

Finally, following the strategy $\mathcal{A}_i$ by the adversary may not lead to the change of the valency of the state, resulting in a 1-valent state $\mathcal{H}_{i+1}$ at the beginning of the next round. Anyway, if the algorithm decides at the very beginning of round $i+1$, the decision cannot be 0. To achieve the $\frac{1}{n^{3/2}}$-strongly-correctness with the decision 0, it must be $\mathbb{P}\mathrm{r}\left(\mathcal{H}_{i+1}, \mathcal{B}\right) < \frac{1}{n^{3/2}}$ for any strategy $\mathcal{B}$. However, the state $\mathcal{H}_{i+1}$ is 1-valent, thus there exists a strategy $\mathcal{B}'$ such that $\mathbb{P}\mathrm{r}\left(\mathcal{H}_{i+1}, \mathcal{B}'\right) > 1 - \frac{1}{n \log n} + \frac{i+1}{n^2}$, which, assuming $i + 1 < n$, excludes decision on value 0. On the other hand, implementing the strategy $\mathcal{A}_i$ from the beginning of round i guarantees that the probability of deciding 1 is at most $\frac{1}{n \log n}$ (which follows from the choice of $\mathcal{A}_i$). Therefore, the adversary can implement $\mathcal{A}_i$ in round $i+1$, then in round $i+2$ and so on. Either it achieves a bivalent state in a round $i + i'$, for some $i' \geq 1$, or in some round it runs out of processes to fail. If the strategy $\mathcal{A}_i$ was continued for i' rounds, then the probability of early deciding, by the union bound, is at most $i' \cdot \frac{1}{n \log n}$. Thus, the lemma is proved. □

We can now finalize the lower bound announced in Theorem 2. For the sake of completeness of this section, we also decided to repeat the statement of the theorem and the proof.

Theorem 7 (Theorem 2)**.** *For any $\left(1 - \frac{1}{n^{3/2}}\right)$-strongly-correct randomized algorithm solving consensus, let T be the random variable denoting the number of rounds until the first process terminates, while R be the random variable denoting the total number of calls to a random source. Then, there is an adversarial strategy that with probability at least $1 - \frac{1}{\log n}$ forces the algorithm to have*

$$T \times (R + T) = \Omega\left(\frac{t^2}{\log n}\right) .$$

Proof. By Lemma 13, the adversary can assign input values such that the initial state is in either a bivalent or a null-valent state. Then, the adversary follows the strategy described in Lemmas 14 and 15, depending whether the current state is null-valent or bivalent. Specifically, if the state is null-valent, the adversary can extend the execution by one more round with probability at least $1 - \frac{1}{n^2}$, by Lemma 14. If the state is bivalent, it can extend the state for some $i' > 1$ rounds with probability at least $1 - \frac{i'}{n \log n}$, such that the new state is again either bivalent or null-valent, or the execution terminates but then the number of failed processes in the previous round would exceed the adversary's limit t. If the algorithm decides to terminate, it must be in either a 0-valent or 1-valent state, since the algorithm is $\left(1 - \frac{1}{n^{3/2}}\right)$-strongly-correct. Therefore, the adversary can prolong the execution either for n rounds or until it runs out of the processes to fail.

Let T be the round in which the execution terminated. If $T = n$, then the theorem follows. Assume then that the adversary stopped implementing its strategy in round T due the fact that in the preceding round it could not fail the desired number of processes. Since the adversary fails at most $16\sqrt{r_i \log n} + 1$ processes in a round i (c.f., Lemma 14 and 15), we obtain

$$t \leq \sum_{i=1}^{T-1} \left(16\sqrt{r_i \log n} + 1\right) \leq 32 \sum_{i=1}^{T-1} \sqrt{(r_i + 1) \log n}\ ,$$

which is equivalent to

$$t^2 \leq 1024 \left(\sum_{i=1}^{T-1} \sqrt{r_i \log n}\right)^2 .$$

Applying the Cauchy-Schwarz inequality to the right hand side of the above, we get

$$t^2 \leq 1024 \left(\sum_{i=1}^{T-1} \sqrt{(r_i + 1) \log n}\right)^2 \leq 1024(T-1) \left(\sum_{i=1}^{T-1} (r_i + 1) \log n\right)\ ,$$

which, after proper rearranging, yields

$$\frac{t^2}{1024 \log n} \leq (T-1) \times (R + T)$$

and proves the theorem. □

D Trading Time for Randomness

Here, we present our algorithm that can trade random calls to a random source for time complexity. The building blocks of the algorithm are these three ideas: (a) similarly to the optimal algorithm from Section B we use the partition of processes into operative and inoperative, rather than distinguishing between faulty and correct, ultimately we first achieve a common decision among the operative ones and then distribute it to the rest; (b) to save on randomness, we partition the set of all processes $\mathcal{P}$ into x groups of size $\lceil \frac{n}{x} \rceil$ each. The high-level idea behind the partition is that achieving a consensus decision, by running the optimal algorithm from Section B, on a group of size $\lceil \frac{n}{x} \rceil$ processes uses less random calls than on the entire network, since the scalability is better than linear we can use this fact to exchange the number of random bits for the running time; (c) to carry over consensus decisions between groups we use round-robin approach. Each group has its time slot in which members of this group try to solve consensus. Then, the members of the group disseminate the consensus decision (if achieved) to other operative processes, who in their turn use this

value as the input for the next consensus calls. Additionally, to achieve consensus conditions with probability 1, processes employ a safety rule analogous to the one in `OptimalOmissionsConsensus`. The pseudocode of the algorithm is given in Algorithm 4. The formal description is given below.

Let $SP_1, \ldots, SP_x$, for an integer $x \in [n]$, be a partition of the set of processes $\mathcal{P}$ into x groups (called also super-processes) of size $\lceil \frac{n}{x} \rceil$ each. Processes compute the partition, based on their identifiers, at the beginning of execution. Also, at the beginning of the execution, the processes compute a graph G of the degree $\Theta(\log n)$ in the same way as in the algorithm described in Section B, i.e., each process uses the result of Theorem 4 to compute its set of neighbors in some predetermined graph G, which is guaranteed to exists by the theorem.

Then the round-robin stage of computation begins. Throughout this stage, processes store their candidate decision value in a variable b, which is initialized to their input value. Also, initially, each process starts the round-robin stage with the operative status set to *true*. This status is maintained based on the number of received messages (messages are send only via edges of the graph G).

The round-robin stage of computation is structured into x phases. In phase i, for $1 \leq i \leq x$, operative processes belonging to the super-process SP_i run a variation of the algorithm OptimalOmissionsConsensus from Section B on their values b. The difference is that the algorithm OptimalOmissionsConsensus is run for a fixed $\Theta\left(\sqrt{x}\log^2 x\right)$ number of rounds, i.e. the OptimalOmissionsConsensus algorithm is terminated at line 16. Although it might not always achieve a consensus decision, a simple observation from the proof of Theorem 5 is that it can happen with a polynomially small probability. On the other hand, we have a control over the number of rounds used for a single run. Processes of other super-processes stay idle for the fixed number of rounds. This is where the "trading" part of the algorithm takes place. The randomness needed to calculate the consensus value on the members of the super-processes is $\Theta\left(\left(\frac{n}{x}\right)^{3/2}\right)$, which scales better than linearly and therefore x separate executions can save random bits compared to the baseline algorithm OptimalOmissionsConsensus from Section B (which is a single execution on the set of n processes).

The outcome of the truncated at line 16 execution of the OptimalOmissionsConsensus algorithm to a process can be two-fold. Either the process got a decision or it has received no messages with a decision when running the OptimalOmissionsConsensus internally (c.f. lines 14-16 in Algorithm 1). In the first case, the process substitutes its candidate value b with the consensus value, in the second case, it substitutes its value b with a default, null-symbol $\perp$. In the last part of the phase, operative processes of the super-process SP_i floods the consensus value (if any) calculated earlier along the graph G. The flooding proceeds in $\Theta(\log n)$ rounds. Each operative process sends a message to its neighbors in G. The message is either empty or contains the consensus decision from the beginning of the phase if the process has already learned this value (operative processes of SP_i know the value from the beginning of the flooding). When receiving messages from neighbors, every process updates the current consensus decision upon receiving, and also keeps track of processes that have not sent any message to it. First, it excludes such processes from further flooding communication in this and any subsequent round-robin phase of the execution. Second, if the number of processes that sent a message drops below $\Delta/3$, the process becomes inoperative and stops its participation in *any* subsequent actions of the algorithm.

The rationale behind the round-robin stage is that processes aim for a phase in which the algorithm OptimalOmissionsConsensus is executed on a super-process whose large fraction of processes is operative. In case like this, with high probability, all processes that remain operative till the end of the execution of OptimalOmissionsConsensus have the same consensus value (and this set is non-empty, assuming that the adversary can crash a small fraction of processes). Then,

Algorithm 4: ParamOmissions

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

$SP_1, \ldots, SP_x \leftarrow$ a partition of $\mathcal{P}$ into x disjoint super-processes of size $\frac{n}{x}$ each;
$\texttt{operative}_p \leftarrow true$;
$V_p \leftarrow V_p \leftarrow$ a set of neighbors of p in a predetermined graph G guaranteed by Theorem 4;
for $i \leftarrow 1$, $i \leq x$ **do**
  if $p \in SP_i$ **then** $\texttt{b}_p, \leftarrow$ OptimalOmissionsConsensus(SP_i, p, b_p) ;
  else wait for a fixed number of $\Theta\left(\sqrt{\frac{n}{x}}\log^2\left(\sqrt{\frac{n}{x}}\right)\right)$ rounds;
  if $p \in SP_i$ **then** $\texttt{consensus\_decision}_p \leftarrow b_p$;
  else $\texttt{consensus\_decision}_p \leftarrow \perp$;
  for $2\log n$ *rounds* **do**
    if $\textit{operative}_p = false$ **then** stay idle until line 25;
    for $q \in V_p$:
      send $\texttt{consensus\_decision}_p$ if q not disregarded before
      receive $\texttt{consensus\_decision}_q$ from q and update:
        $\texttt{consensus\_decision}_p \leftarrow \texttt{consensus\_decision}_p \cup \texttt{consensus\_decision}_q$;
      if q has not sent a message, disregard sending to q in any future round;
    if *number of received messages is less than* $\Delta/3$ **then**
      $\texttt{operative}_p \leftarrow false$;
  end
  $b_p \leftarrow \texttt{consensus\_decision}_p$;
end
/* implementation of a safety rule */
$\texttt{decide} \leftarrow false$;
if $\textit{operative}_p = true$ **then**
  send b_p to all processes in $\mathcal{P}$;
  receive bits sent in the previous round; let variables $\texttt{ones}_p, \texttt{zeros}_p$ denote the number of 1's and 0's received;
  if $\textit{ones}_p > \frac{18}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $b_p \leftarrow 1$;
  else if $\textit{ones}_p < \frac{15}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $b_p \leftarrow 0$;
  if $\textit{ones}_p > \frac{27}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $\texttt{decided}_p \leftarrow true$;
  else if $\textit{ones}_p < \frac{3}{30}(\textit{ones}_p + \textit{zeros}_p)$ **then** $\texttt{decided}_p \leftarrow true$;
end
if $\textit{operative}_p = true$ *and* $\textit{decided}_p = true$ **then send** b_p to all processes in $\mathcal{P}$;
else if *any message* b_q *received from some process* q **then** $b_p \leftarrow b_q$;
/* in the above, q can be chosen arbitrarily from the received messages */
if $\textit{decided}_p = true$ *or* ($\textit{operative}_p = false$ *and* p *received a message in the previous round*) **then decide** b_p;
else
  if $\textit{operative}_p = true$ **then** p takes part in any deterministic synchronous consensus algorithm (e.g., [14]); if p reaches agreement in that protocol, it broadcast the decision to all processes in $\mathcal{P}$ and it decides on the algorithm's decision;
  else p remains idle until a decision is sent to it; upon receiving a decision it decided on this value;
end

the flooding procedure on the random graph guarantees that the consensus value is propagated to all processes that are operative. At this point, there is only one value in the system, so it can be expected that this value will remain the consensus decision regardless of the adversarial strategy. Since in the above scenario there is a $O(1/n)$ chance that the operative processes fail to reach consensus, after the round-robin stage ends, the algorithm deploys a safety rule, cf. lines 15-30. Specifically, the operative processes count the number of 1's and 0's among themselves and in the rare case that both these numbers are close to each other, they execute the deterministic protocol from [14].

Below we present the theorem summarizing the ParamOmissions algorithm.

Theorem 8. *The algorithm* ParamOmissionsConsensus *solves consensus (with probability* 1*) against the adaptive adversary that can control* $t < \frac{n}{60}$ *processes. For any integer parameter* $x \in [n]$*, it achieves the following bounds, with high probability: it terminates in* $\tilde{O}\left(\sqrt{xn}\right)$ *rounds, uses* $\tilde{O}\left(n\sqrt{\frac{n}{x}}\right)$ *random bits and* $\tilde{O}\left(n \cdot \frac{n}{x} + xn + n^2\right)$ *bits of communication.*

Note, that if we denote by R the number of random bits used by the algorithm, then by rearranging the remaining terms in the above theorem accordingly, we get the formulation of Theorem 3.

Analysis. Processes can become inoperative only by receiving less than $\Delta/3$ messages from their neighbors in the graph G. This is a milder requirement compared to the one used in the algorithm OptimalOmissionsConsensus, where not only the process has to receive $\log n$ messages in the graph G, but it also has to remain operative during other sub-protocols. Also, we allow at most $\frac{n}{40}$ failures which are twice as small as in the proof of Theorem 5, thus the following lemma is a consequence of Lemma 7 proven in Section B.2.

Lemma 16. *With high probability, at least* $n - 3t \geq \frac{57n}{60}$ *processes stay operative throughout the execution of the algorithm.*

We say that a super-process SP_i is *reliable* if at least $\frac{29}{30}\frac{n}{x}$ processes belonging to it remain *non-faulty* to the end of the execution and at least one of these process is *operative* throughout the execution.

Lemma 17. *In any execution, there is at least one reliable super-process with high probability.*

Proof. Since there are at most $\frac{n}{60}$ faulty processes, thus by the pigeonhole principle there are at least $\frac{x}{2}$ super-processes that contain at most $\frac{1}{20}\frac{n}{x}$ faulty processes. Consequently, at least $\frac{1}{2}\frac{29}{30} \cdot n$ processes belonging to these super-processes remain non-faulty throughout the entire execution. By Lemma 16, there is another set of at least $n - 3t \geq \frac{57n}{60}$ processes that are always operative. Since $\frac{57}{60} + \frac{1}{2}\frac{29}{30} > 1$, thus these two sets have a non-empty intersection, proving that there is a super-processes that has at most $\frac{1}{30}\frac{n}{x}$ faulty members and among the non-faulty members at least one is operative to the end of the algorithm. □

Next, we recall a property of the flooding algorithm (e.g. Lemma 8) used in ParamOmissions that was first used to prove Theorem 5. Note, that that the flooding is designed in the same way as the GroupBitsSpreading algorithm used in OptimalOmissionsConsensus (cf. Algorithm 3), only the content of relayed messages is different. Therefore, by repeating the reasoning used in Lemma 6, we obtain the following.

Lemma 18. *For any two processes,* p, q *that are operative at the end of a phase* i*, the consensus decision of* p *(if different than* $\perp$*) has been relayed to* q *in the flooding part of the phase* i *and vice-versa, the consensus decision of* q *(if different than* $\perp$*) has been relayed to* p*.*

Finally, we observe that the same arguments used for proving the correctness of the stopping rule in the algorithm OptimalOmissionsConsensus apply to the stopping rule used in line 15- 30 of the ParamOmissions algorithm. Thus we get the following.

Lemma 19. *With probability* 1*, all non-faulty processes decide and the decision is on the same value.*

Next, we proceed to the proof of the main theorem of the section.

Proof of Theorem 8. The agreement and termination follow from Lemma 19. The validity follows from the fact that the truncated version of OptimalOmissionsConsensus also satisfies validity, cf. the proof of Theorem 5.

Next, we analyze the complexity measures. Denote SP_i the reliable super-process which, by Lemma 17, exists with high probability. Since at least $\frac{29}{30}\frac{n}{x}$ processes belonging to SP_i remain non-faulty, we can apply Theorem 5 and conclude that the truncated execution of the algorithm OptimalOmissionsConsensus on the group SP_i in the appropriate phase of the round-robin scheme results in all non-faulty processes of SP_i having the same current consensus decision b with high probability. By the reliability of the super-process, among these non-faulty processes there exists at least one operative process. Thus, in the flooding part of the round-robin phase i, all other operative processes acquire the output of the OptimalOmissionsConsensus from the beginning of the phase, by Lemma 18. Therefore, after the flooding ends, all operative processes have the same value b_p, see line 13. Since in any phase, only operative processes execute the OptimalOmissionsConsensus algorithm (see line 10), each subsequent run of the algorithm has the property that all parties begin with the same input bit.

We observe that in case when all processes start with the same input bit, processes that return a value at the end of the algorithm OptimalOmissionsConsensus can only return the value being the input bit. This follows from analyzing lines 9-13 in the algorithm OptimalOmissionsConsensus (Algorithm 1). If there is only one input value in the system, no process will access the random coin and only the input value will be manipulated to the end of the algorithm. It follows that after the round-robin phase performed on the reliable super-process SP_i, all operative processes have the same value in the variable b_p. In this case, in line 18 all operative processes receive only one bit and thus they always set the variable decided to *true* when executing conditional statements in lines 21-22. In consequence, all operative processes spread the same decision in line 24. Since the number of operative processes is at least $\frac{57n}{60} > \frac{n}{60} = t$, every non-faulty process eventually receives a decision and decides on it in line 26. Observe also that in this case no operative process has its variable `decided` set to false and thus the deterministic protocol in line 28 will not be executed.

The derivation of the complexity measures comes directly from conditioning on the event that all operative processes end having the variable `decided` set to *true*, which from now on we assume.

As for the running time, every phase lasts $\tilde{O}\left(\sqrt{\frac{n}{x}}\right)$ rounds, because it consists of a single run of the truncated algorithm OptimalOmissionsConsensus, which time complexity can be easily derived from Theorem 5, and of $2\log n$ rounds of flooding. There are x phases and two additional rounds after the round-robin phases end, thus the bound on the time complexity follows with high probability.

We bound the communication bit complexity analogously. Each phase uses $O\left(\left(\frac{n}{x}\right)^2 \log^4\left(\frac{n}{x}\right) + n\log^2 n\right)$ communication bits. The first term corresponds to the execution of the algorithm OptimalOmissionsConsensus on a set of size $\frac{n}{x}$ (c.f., Theorem 5), the latter corresponds to $4\log n$ rounds of flooding. Observe that in the flooding, in each round every

process sends at most $\Theta(\log n)$ messages of size $O(1)$. Multiplying the phase bit complexity by the number of phases and adding $O(n^2)$ term, corresponding to the last two rounds, explains the bound on the bit complexity of the whole algorithm and that it holds whp.

The random bits are spent only in the runs of OptimalOmissionsConsensus. There are at most x independent runs of the algorithm, each on a set of processes of size $\frac{n}{x}$. Applying Theorem 5 yields that the total number of used random bits is $O\left(x \cdot \left(\frac{n}{x}\right)^{3/2} \log^2\left(\frac{n}{x}\right)\right) = \tilde{O}\left(n\sqrt{\frac{n}{x}}\right)$, which completes the proof of the theorem. □