Fault-Tolerant Consensus in Quantum Networks

MohammadTaghi Hajiaghayi [*]   Dariusz R. Kowalski [†]   Jan Olkowski [*]

Abstract

Fault-tolerant *consensus* is about reaching agreement on some of the input values in a limited time by non-faulty autonomous processes, despite of failures of processes or communication medium. This problem is particularly challenging and costly against an adaptive adversary with full information. Bar-Joseph and Ben-Or [8] (PODC'98) were the first who proved an absolute lower bound $\Omega(\sqrt{n/\log n})$ on expected time complexity of consensus in any classic (i.e., randomized or deterministic) message-passing network with n processes succeeding with probability 1 against such a strong adaptive adversary crashing processes.

Seminal work of Ben-Or and Hassidim [10] (STOC'05) broke the $\Omega(\sqrt{n/\log n})$ barrier for consensus in classic (deterministic and randomized) networks by employing quantum computing. They showed an (expected) constant-time quantum algorithm for a linear number of crashes $t < n/3$.

In this paper, we improve upon that seminal work by reducing the number of quantum and communication bits to an arbitrarily small polynomial, and even more, to a polylogarithmic number – though, the latter in the cost of a slightly larger polylogarithmic time (still exponentially smaller than the time lower bound $\Omega(\sqrt{n/\log n})$ for classic computation).

Keywords: distributed algorithms, quantum algorithms, adaptive adversary, crash failures, Consensus, quantum common coin, approximate counting

1 Introduction

Consensus is about making a common decision among the processes' initial input values in a limited time by every non-faulty process, despite the faulty behaviour of some of the players. Since its introduction by Pease, Shostak and Lamport [31] (JACM'80), who ruled out trivial solutions (such as always deciding on the same bit), fault-tolerant consensus has constantly been among foundation problems in distributed computing. This problem has been studied in synchronous/asynchronous and deterministic/randomized computation models and under various fault-tolerant or adversarial models: Fail-Stop (crashes) and Byzantine, Static and Adaptive, Computationally Bounded and Unbounded Adversaries - just to name a few (see Section 2.1 for related work).

While the landscape of Consensus problem under the classic model of computation is well-developed, much less is known if we allow for quantum computation. The seminal work of Ben-Or and Hassidim [10] (albeit a short 5-pager in STOC'05) broke the $\Omega(\sqrt{n/\log n})$ rounds time barrier for classic computing by employing quantum computing to further use the power of randomization in distributed computing. They showed an (expected) constant-time quantum algorithm for a linear number of crashes $t < n/3$, however, the algorithm is inefficient in terms of communication bits and, even more importantly, in terms of the number of quantum bits (qubits), as it uses $\Omega(n)$ of them per process. Since then no algorithm has managed to reduce the quantum resources needed to solve Consensus. Because generating, maintaining and sending quantum bits is extremely costly (today's quantum devices use less than 500 qubits), thus the main question of our paper emerges naturally:

Could the number of qubits be substantially reduced without harming the time complexity?

Distributed setting. We consider a quantum synchronous message-passing model (c.f., [7]), consisting of n synchronous processes (also called players), each with common clock (a clock tick is called a round or a step) and unique id from the known set $\mathcal{P} = [n] = \{1, \ldots, n\}$.

Between any pair of processes we assume the existence of a quantum channel being able to transmit reliable* messages caring quantum bits, qubits. For the sake of completeness, we also augment the model by classic point-to-point channels between any pair of processes. In each round, a process can send (personalized) quantum and classic messages to any selected subset of other processes. After *multicasting messages*, in the same round a process *receives messages* that were just sent to it by other processes, and performs *local computation*, which involves both quantum and classic bits.†

Processes are prone to *crash failures*, also called *fail-stops*. A crashed process permanently stops any activity, including sending and receiving messages.

We model crashes as incurred by a full-information *adversary* (the same as in [8, 10]) that knows the algorithm, the exact pure quantum state (see Section 3) and the classic state of the system at any point of an execution, and has an unbounded computational power. The adversary decides which processes to fail and when. The adversary is also *adaptive* – it can make a decision on-line based on its current full-knowledge of the system. However, the adversary does not know the future computation, which means that it does not know future random bits drawn by processes.

As for the quantum part, the adversary can apply no quantum operation to the system, but it is aware of all quantum and classic changes of state that the network undergoes. If a process is crashed by the adversary, we assume that its quantum bits are not destroyed (in particular, entangled qubits in other processes do not collapse but maintain their entanglement), however they cannot be used in further computation.

---

*Messages are not lost nor corrupted while in transit.

†Local computation also decides what messages to send in the next round and to whom.

Failures are *not clean* – when a process crashes when attempting to multicast a message, then some of the recipients may receive the message and some may not; this aspect is controlled by the adversary. A *t-adversary* is additionally restricted by the the number of crashed processes being *smaller than t*; if $t = n$ then the *n*-adversary is also called an unbounded adversary (note that even in such case, at least one process must survive for consensus to make sense). Throughout the paper, we will be calling the adversary described above "adaptive", for short.

***Consensus* problem:** each process p has its own initial value $input_p$ and has to output a (common) decision value, so that the following conditions hold: *validity* – decision must be on an input value of some process; *agreement* – no two processes decide on different values; and *termination* – each process eventually decides on some value, unless it is faulty. All those three requirements must hold with probability 1. We focus on *binary consensus*, in which initial values are in $\{0, 1\}$.

Correctness and complexity – in terms of time (the number of rounds needed for all processes to terminate) and the number of quantum bits (qubits) and communication bits – are analyzed and maximized (worst-case) against an adaptive adversary.

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 sufficiently large positive constant c by linear scaling of parameters.

2 Our Results

In this work, we focus on improving quantum bits' and communication complexities (without harming time complexity) of quantum algorithms solving Consensus problem with probability 1 against an adaptive full-information adversary capable of causing processes' crashes. We observe that the maximum, per process, number of communication bits in Consensus problem is $\Omega(n)$, therefore one can only hope to improve amortized communication complexity (per process), see Lemma 6 in Appendix D.

Our first main result is a quantum algorithm that solves Consensus (correctness with probability 1) in expected constant number of rounds and amortized number of qubits and communication bits per process being an arbitrarily low polynomial. This directly improves, by a polynomial factor, the result of Ben-Or and Hassidim [10], which required a super-linear number of qubits and communication bits, amortized per process. More precisely, in Section 4 we give an algorithm and in Section A we prove the following result (see also its technical counterpart – Theorem 6):

Theorem 1. *For any $\epsilon > 0$, there is an algorithm solving consensus against an adaptive $n/3$-adversary in expected $O(1)$ rounds while using $O(n^\epsilon)$ qubits and communication bits (amortized) per process, whp.*

Although our algorithm, called CHEAPQUANTUMCONSENSUS, uses an idea of so called weak global coin analogously to e.g., [19, 10], ours is based on two entirely different qubit-and-communication efficient algorithmic techniques. First technique is a new quantum implementation of a weak global coin, c.f., Definition 1, called CHEAPQUANTUMCOIN. It works in constant, $O\left(\frac{1}{\epsilon}\right)$, time, uses an arbitrarily small polynomial number of bits $O(n^\epsilon)$, amortized per process (for any chosen constant $\epsilon > 0$), and results in all non-faulty processes returning the same value with a non-zero constant probability, c.f., Theorem 3. Second is a deterministic algorithm for counting, which returns, in constant time, a count of the number of input 1 (or 0, resp.) among the processes. The output could be different at different processes, due to failures and a short time for counting, but each of them is a number not smaller than the number of processes with input 1 (0, resp.) at the end of the procedure and not larger than the number of processes with input 1 (0, resp.) in the beginning of

the procedure. It uses only an arbitrary small polynomial number of communication bits, amortized per process. We give an overview of these two techniques in the next Sections 5 and 6, respectively, and their analysis in Sections B and C, respectively.

Although constant-time algorithms cannot achieve low communication, as we argue in Lemma 7 in Appendix D, interestingly, we show that our main algorithm could be re-instantiated in such a way that it uses only a polylogarithmic number of qubits and communication to solve consensus in a slightly (polylogarithmically) larger number of rounds (see Section 4 and Appendix A, and technical counterpart Theorem 7 for more details):

Theorem 2. *There is an algorithm solving consensus against an unbounded adaptive adversary in polylogarithmic number of rounds, in expectation, while using a polylogarithmic number of qubits and communication bits (amortized) per process, whp.*

We believe that the newly developed techniques could be also applied to other types of failures, after failure-specific modifications. For example, although message omission failures require linear amortized communication per process (c.f., [24]), one could still use a small polynomial or even a polylogarithmic number of qubits (together with a linear number of classic communication bits) per process, if qubits are handled according to our techniques while some additional classic communication bits are introduced to handle message omissions. We leave details to follow-up work.

2.1 Previous and Related Work

Consensus in the classic (non-quantum) model. Bar-Joseph and Ben-Or [8] (see also their extended version [9]) proved a lower bound $O(\frac{\sqrt{n}}{\log n})$ on expected time complexity of consensus against an adaptive adversary. They also complemented it by time-optimal randomized algorithm. Their algorithm uses expected $O(\frac{n^{3/2}}{\log n})$ number of communications bits, amortized per process, which has been recently improved by Hajiaghay, Kowalski and Olkowski [25] to $O(\sqrt{n})$ (while maintaining the almost-optimal round complexity $O(\sqrt{n}\log n)$).

Fisher and Lynch [20] proved a lower bound $f+1$ on *deterministic* consensus with f crashes (that actually occurred, i.e., $f < t$), thus separating deterministic solutions from randomized. Regarding communication complexity, Amdur, Weber and Hadzilacos [3] showed that the amortized number of messages per process is at least constant, even in some failure-free execution. Dwork, Halpern and Waarts [18] found a solution with $O(\log n)$ messages per process, but requiring an exponential time, and later Galil, Mayer and Yung [22] developed a message-optimal algorithm working in over-linear $O(n^{1+\varepsilon})$ time, for any $0 < \varepsilon < 1$. They also improved the communication further to a constant number of communication bits per process, but the resulting algorithm was exponential in the number of rounds. Chlebus and Kowalski [12] showed that consensus can be solved in $O(f+1)$ time and with $O(\log^2 f)$ messages if only the number $n-f$ of non-faulty processors satisfies $n-f = \Omega(n)$. It was later improved in [13] to $O(f+1)$ time and $O(\text{polylog } n)$ number of communication bits. All the abovementioned communication complexities are amortized per process.

Quantum consensus. To the best of our knowledge, almost all previous papers on quantum consensus concentrated on assuring feasibility of the problem against strong Byzantine adversaries, c.f., [15, 26, 28], or on optimizing time complexity, including the work of Ben-Or and Hassidim [10] achieving constant time against an adaptive adversary. Sparse quantum communication has been considered by Chlebus, Kowalski and Strojnowski in [14], but their protocols work correctly only with some probability smaller than 1 and for a specific number of failures corresponding to the probability of success. Another difference is that they used quantum operations to encode the

classical inputs in quantum registers and used it to directly solve consensus. In this paper, we show another, more-efficient approach, in which we first create a quantum, weak global coin and later employ this tool to the state-of-the-art framework of solving consensus based on the common coin. Other distributed computing problems, not necessarily fault-prone, were also analyzed in quantum models, c.f., [11, 32, 4, 33].

More efficient classic randomized solutions against *weak adversaries*. Whenever weaker oblivious adversaries are considered, randomness itself occurred to be enough in reducing time complexity to a constant. Chor, Merritt and Shmoys [16] developed constant-time algorithms for consensus against an *oblivious adversary* – that is, the adversary who knows the algorithm but has to decide which process fails and when before the execution starts. Their solution, however, requires a linear number of communication bits per process. Gilbert and Kowalski [23] presented a randomized consensus algorithm that achieves optimal communication complexity of $O(1)$ amortized communication bits per process while terminating in $O(\log n)$ time with high probability (whp for short), as long as the number of crashes $f < n/2$.

Classic consensus in more demanding models. Dolev and Reischuk [17] and Hadzilacos and Halpern [24] proved the $\Omega(f)$ lower bound on the amortized message complexity per process of deterministic consensus for *omission or (authenticated) Byzantine failures*. However, under some limitation on the adversary and requiring termination only whp, the sublinear expected communication complexity $O(\sqrt{n} \text{ polylog } n)$ per process can be achieved even in case of Byzantine failures, as proved by King and Saia [27]. Such limitations are apparently necessary to achieve subquadratic time complexity for Byzantine failures, c.f., Abraham et al. [1]. *Asynchrony* also implies large communication – Aspnes [5] proved a lower bound $\Omega(n/\log^2 n)$ on communication complexity per process. The complexity bounds in this setting have been later improved, c.f., [6, 2].

3   Technical preliminaries

Quantum model of computation. We focus on quantum properties relevant to our algorithms. A quantum system operates on qubits. Single qubit can be either in a pure or a mixed state. A pure state is a vector in a 2-dimensional Hilbert space $\mathcal{H}$, while a mixed state is modelled as a probabilistic distribution over pure states. Similarly, a register consisting of d qubits can also be in a pure or a mixed state. A pure quantum state, denoted $|x\rangle$, is a vector of 2^d-dimensional Hilbert space $\mathcal{H}^{\otimes d} = \mathcal{H} \otimes \ldots \otimes \mathcal{H}$. Again, a mixed state of larger dimension is viewed a probabilistic distribution over pure states. In our paper, we will operate only on pure states, and we will use only the standard computational basis of the Hilbert space, which consists of vectors $\{|b_1 \ldots b_d\rangle : b_1 \ldots, b_d \in \{0,1\}^d\}$, to describe the system. Therefore, any state $|x\rangle$ can be expressed as $|x\rangle = \sum_{i=0}^{2^d-1} \alpha_i |i\rangle$, with the condition that $\sum_i |\alpha_i|^2 = 1$, since quantum states can be only normalized vectors.

Transitions, or equivalently – changes of states of a quantum system, are given by unitary transformations on the Hilbert space of d qubits. These unitary transformations are called *quantum gates*. These operations are exhaustive in the sense that any quantum computation can be expressed as a unitary operator on some Hilbert space. There are small-size sets of quantum gates working on two-dimensional space that are universal – any unitary transformation on a 2^d-dimensional quantum space can be approximated by a finite collection of these universal gates. In our applications, any quantum algorithm computation run by a process requires polynomial (in n) number of universal gates.

4

Finally, an important part of quantum computation is also a quantum measurement. Measurements are performed with respect to a basis of the Hilbert space – in our case this is always the computational basis. A complete measurement in the computational basis executed on a state $|x\rangle = \sum_{i=0}^{2^d-1} \alpha_i |i\rangle$ leaves the state in one of the basis vectors, $|i\rangle$, for $i \in \{0,1\}^d$, with probability α_i^2. The outcome of the measurement is a classic register of d bits, informing to which vector the state has been transformed. It is also possible to measure only some qubits of the system, which is called a partial measurement. If A describes the subset of qubits that we want to measure and B is the remaining part of the system, then the partial measurement is defined by the set of projectors $\{\Pi_i = |i\rangle_A \langle i|_A \otimes I_B \mid \text{for } i \in \{0,1\}^d\}$.[‡] In the former, a subscript refers to the part of the system on which the object exists, I denotes the identity function, while $\langle i|$ is a functional of the dual space to the original Hilbert space (its matrix representation is the conjugate transpose of the matrix representation of $|i\rangle$). If before the measurement the system was in a state $|x\rangle_{AB}$ then, after the measurement, it is in one of the states $\{\Pi_i |x\rangle_{AB} \mid \text{for } i \in \{0,1\}^d\}$, where state $\Pi_i |x\rangle_{AB}$ is achieved with probability $\langle x|_{AB} \Pi_i |x\rangle_{AB}$.[§] The reader can find a comprehensive introduction to quantum computing in [29].

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

The following combinatorial properties are of utter importance in the analysis of our algorithms. Graph G is said to be *ℓ-expanding*, or to be an *ℓ-expander*, if any two subsets of ℓ nodes each are connected by an edge. Graph G is said to be *(ℓ, α, β)-edge-dense* if, for any set $X \subseteq V$ of *at least ℓ nodes*, there are at least $\alpha|X|$ edges internal for X, and for any set $Y \subseteq V$ of *at most ℓ nodes*, there are at most $\beta|Y|$ edges internal for Y. Graph G is said to be *$(\ell, \varepsilon, \delta)$-compact* if, for any set $B \subseteq V$ of at least ℓ nodes, there is a subset $C \subseteq B$ of at least $\varepsilon\ell$ nodes such that each node's degree in $G|_C$ is at least δ. We call any such set C a *survival set for B*.

4 Consensus Algorithm

On a very high level, our consensus algorithm CHEAPQUANTUMCONSENSUS repeatedly uses the counting procedure FASTCOUNTING, specified in Section 6, to compute the number of 0's and 1's preferred by processes, see line 3. (Recall that the outcomes of FASTCOUNTING could be slightly different across processes, but by no more than the number of crashes.) Depending on the outcome, each process may change its preferred value to the dominating one (among the received preferred values), decide on it if the domination is substantial, or run the quantum common coin procedure in some almost-unbiased cases – see lines 14-17 in the pseudocode of CHEAPQUANTUMCONSENSUS in Figure 1; all lines involving communication are underlined. The latter is a well-established general

---

[‡]Whenever it could be irrelevant, from the context, we may follow the standard notation in quantum computing and skip writing normalizing factors.

[§]$\Pi_i |x\rangle_{AB}$ and $\langle x|_{AB} \Pi_i |x\rangle_{AB}$ are simply linear operations on matrices and vectors.

framework for solving consensus, proposed first by Bar-Joseph and Ben-Or [9] in the context of classic randomized computation against an adaptive adversary. Our novelty is in two new techniques, employed in lines 3 and 18 of the pseudocode in Figure 1: fast and quantum/communication-efficient counting and weak global coin, respectively. Both these techniques use parameters x, d, α, which, roughly speaking, correspond to the density of random communication graphs used in these algorithms. The detailed performance formulas of these algorithms, with respect to those parameters, are stated in Theorems 3 and 5.

In the heart of these two techniques lies a crucial observation: consensus (as well as common coin and counting) could be achieved quickly even if many processes do not directly exchange messages, but use some carefully selected sparse set of communication links instead. This way, instead of creating qubits for each pair of processes, we could do it only per some pairs corresponding to some communication links to be used. Obviously, this set of links, modeled as an evolving communication graph, needs to be maintained adaptively and locally by processes throughout the execution – otherwise, an adaptive adversary would learn it and design crashes to separate processes and prevent consensus.

Algorithm 1: CHEAPQUANTUMCONSENSUS for process p

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

1   $b_p \leftarrow input_p$; $r \leftarrow 1$;   decided $\leftarrow FALSE$;

2   **while** $TRUE$ **do**

3   participate in FASTCOUNTING$(\mathcal{P}, p, b_p)$ (run with parameters $x = n^\epsilon, d = \log n, \alpha = n^\epsilon$) that counts the processes who have $b_p = 1$ and the processes who have $b_p = 0$; let O_p^r, Z_p^r be the numbers of ones and zeros (resp.) returned by FASTCOUNTING;

4      $N_p^r \leftarrow Z_p^r + O_p^r$;

5   **if** $(N_p^r < \sqrt{n/\log n})$ **then**

6   1) send b_p to all processes; 2) receive all messages sent to p in round $r + 1$;

7   3) implement a deterministic protocol for $\sqrt{n/\log n}$ rounds;

8   **end**

9   **if** $decided = TRUE$ **then**

10   diff $\leftarrow N_p^{r-3}$ - N_p^r;

11   **if** $(diff \leq N_p^{r-2}/10)$ **then** STOP;

12   **else** decided $\leftarrow FALSE$;

13   **end**

14   **if** $O_p^r > (7N_p^r - 1)/10$ **then** $b_p \leftarrow 1$, decided $\leftarrow TRUE$;

15   **else if** $O_p^r > (6N_p^r - 1)/10$ **then** $b_p \leftarrow 1$;

16   **else if** $O_p^r < (4N_p^r - 1)/10$ **then** $b_p \leftarrow 0$, decided $\leftarrow TRUE$;

17   **else if** $O_p^r < (5N_p^r - 1)/10$ **then** $b_p \leftarrow 0$;

18   **else** set b_p to the output of CHEAPQUANTUMCOIN$(\mathcal{P}, p)$ executed with parameters $d = \log n, \alpha = n^\epsilon$;

19      $r \leftarrow r + 1$;

20   **end**

21   **return** b_p ;   /* p outputs final decision */

Algorithm's description. Each process p stores its current choice in b_p (which is initialized to p's input). The value b_p in the end of the algorithm indicates p's decision. Now, processes use $O(1)$ (in expectation) *phases* to update their values b_p such that eventually every process keeps the same

decision. To do so, in a round r every process p calculates the number of processes whose current choice is 1 and the number of processes whose current choice is 0, denoted O_p^r and Z_p^r respectively. Based on these numbers, process p: either sets b_p to 1, if the number O_p^r is large enough; or it sets b_p to 0, if the number Z_p^r is large; or it replaces b_p with a random bit, if the numbers of zeros and ones are close to each other. If for generating the random bit, in line 18, processes use a quantum implementation of a weak global coin (implemented with CHEAPQUANTUMCOIN algorithm, specified in Section 5), they will all have the same value b_p with constant probability unless more than third of alive processes crash. Assuming the presence of the adaptive adversary, this could not be achieved quickly if using classic communication only. Once it happens with the help of the quantum weak global coin, the conditional statements in lines 14-17, run in the next iteration of the "while" loop, guarantee that once the majority of processes have the same value b_p, the system converges to this value in at most 2 phases. Since the probability of this event is constant (guaranteed by the quantum weak global coin combined with the analysis of the first classic framework in [9]), the expected number of phases before the consensus algorithm terminates is constant. That reasoning holds, assuming that at most $1/3$ fraction of processes crashed (we will generalize it to any $t \leq n$ at the end of this section).

As mentioned earlier, the major improvement in the above protocol comes from using novel techniques for counting and weak global coin. For the former, we use the FASTCOUNTING algorithm, which, with the choice of parameters given in line 3, works in $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds and uses $O(n^{1+3\epsilon} \log^2 n)$ (classic) communication bits in total. Similarly, the CHEAPQUANTUMCOIN algorithm, executed in line 18, terminates in $O\left(\left(\frac{1}{\epsilon}\right)^3\right)$ rounds and uses $O\left(n^{1+2\epsilon} \log^2 n\right)$ both quantum and classic bits; we need to divide the communication formulas by n to obtain the complexity amortized per process. By applying these two techniques in the fashion described above, we get Theorem 1; detail proof is deferred to Appendix A.

Handling arbitrary number of crashes. Consider $O(\log n)$ repetitions of the main loop (phases) of the CHEAPQUANTUMCONSENSUS algorithm. If during these phases, the processes with value $b_p = 1$ become a large majority (at least $\frac{6}{10}$ fraction of alive processes), then, as discussed before, every process will decide within the next two rounds. The same holds if processes with value $b_p = 0$ start to overpopulate by a ratio of $\frac{6}{10}$ all non-faulty processes. On the other hand, if the cardinalities of the two groups with different values b_p are close to each other, then the processes execute the CHEAPQUANTUMCOIN algorithm. It outputs a random bit (the same in every participating process), under the assumption that at least a $\frac{2}{3}$ fraction of processes that started this phase as non-faulty have not crashed during this phase. However, in these $O(\log n)$ phases there must be at least one phase in which the property of a $\frac{2}{3}$ fraction of processes survive holds. In what follows, we argue that if the adversary can crash arbitrarily many processes, but smaller than n, then the expected number of phases should still be $O(\log n)$. Now, to obtain the algorithm stated in Theorem 2, we make two more adjustments of the original CHEAPQUANTUMCONSENSUS algorithm. In lines 3 and 18, processes execute the algorithms FASTCOUNTING and CHEAPQUANTUMCOIN, respectively, with parameters x, d, α set as follows: $x = 2, d = \log n, \alpha = \log n$. This corresponds to a use of sparse graph for communication (of degree roughly $O(\log n)$). In consequence, the time complexity of the FASTCOUNTING algorithm increases to $O(\log^4 n)$, but the communication complexity decreases to $O(\log^8 n)$ amortized per process. The details are presented in Section 6, and performance follows from putting the abovementioned parameters to the formulas in Theorem 5). Similarly, the time complexity of the CHEAPQUANTUMCOIN algorithm increases to $O(\log^3 n)$, but the communication complexity (both quantum and classical) decreases to $O(\log^7 n)$ amortized per process. The details are presented in Section 5, and performance follows from putting the abovementioned parameters

to the formulas in Theorem 3).

5   Qubit-and-Communication Efficient Quantum Common Coin

In this section, we design a new t-resilient weak global coin, for $t < n$, with the help of quantum communication and computation. The coin must satisfy the following definition:

Definition 1 ([10])**.** *Let $\mathcal{C}$ be a protocol for n players (with no input), where each player i outputs a (classical) bit $v_i \in \{0,1\}$. We say that the protocol $\mathcal{C}$ is a t-resilient weak global coin protocol (or computes a weak global coin, for short) with fairness $\rho > 0$, if for any adaptive t-adversary and any value $b \in \{0,1\}$, with probability at least ρ, $v_i = b$ for all good players i.*

On the high level, our protocol CHEAPQUANTUMCOIN chooses a leader process uniformly at random and all other processes agree on the random bit proposed by the leader. Quantum phenomena is used to hide the random choices of the leader and its output from the adaptive adversary when processes communicate with each other. The idea was first proposed in [10], yet there are key differences between that work and our algorithm. Instead of all-to-all communication, which required large number of qubits, we use a sequence of random graphs of node degrees $d, d\alpha^1, \ldots, d\alpha^k$, respectively, where $d, \alpha \in \Omega(\log n)$ and $k = \lceil \log n / \log \alpha \rceil$ are some integer parameters. The vertices of these graphs correspond to processes and edges correspond to communication link – each process communicates with neighbors in one of the graphs at a time. If the graph is chosen properly (i.e., so that there is no situation in which too many processes use denser graphs), it reduces the communication complexity, but simultaneously imposes a new challenge. Mainly, the communication procedure has to now assure delivery of quantum bits between every two non-faulty processes regardless of the pattern of crashes. For instance, if only one random graph of degree d was used then the adversary could easily isolate any vertex using only $O(d)$ crashes (i.e., by crashing all its neighbors). Hence, strictly speaking, assuring such delivery is not possible while using a sparse communication graph as relays, but we show that a certain majority could still come up with a common coin value based only on their exchanges with neighbors in the communication graphs; they could later propagate their common value to other processes by adaptively controlling their (increasing) set of neighbors, taken from subsequent communication graphs of increasing density. A thorough analysis shows that in this way it is possible to achieve the same quantum properties that are guaranteed by Ben-Or's and Hassidim's global coin [10], and at the same time reducing the quantum communication by a polynomial factor. Formally, we prove the following result.

Algorithm's description.   We now describe the CHEAPQUANTUMCOIN algorithm. Its pseudocode is presented in Figure 2. It takes as an input: a process name p and two integers, d and α. The two latter parameters are used to determine communication pattern between non-faulty processes, and their choice determines complexity of the algorithm.

As mentioned before, processes use quantum equivalent of a procedure in which processes draw random names from the range $[1, \ldots, n^3]$ and decide on a random bit proposed by the process with the largest name.[¶] We view the quantum part of the algorithm as a quantum circuit on the joint space of all qubits ever used by different processes. Due to the distributed nature of the system, not all quantum operations are allowed by the quantum circuit. That is, (i) any process can perform arbitrary unitary gates on its part of the system, (ii) unitary gates on qubits of different processes might be performed, but must be preceded by quantum communication that send qubits involved in the operation to a single process. The communication can be either direct or via relays. We

---

[¶]Note that the latter procedure cannot be used against an adaptive adversary, as it could crash such a leader.

next explain what unitary gates can be used to simulate the classic algorithm of the leader election in the quantum distributed setting. For the purpose of explanation, we assume that the qubits needed to execute each of the following gates have been delivered to proper processes. The pattern of communication assuring this delivery is described in the next paragraph.

1. **Hadamard gate.** This unitary operation on a single qubit is given by the matrix $H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. When applied to every qubit of an m-qubit state $|0\ldots0\rangle$, it produces a uniform superposition of vectors of the computational basis, i.e.,

$$H^{\otimes m}|0\ldots0\rangle = \frac{1}{\sqrt{2}\cdot n^{3/2}} \sum_{\ell_1,\ldots,\ell_{m-1}\in\{0,1\}^{m-1}, b\in\{0,1\}} |\ell_1\ldots\ell_m b\rangle = \frac{1}{2^{m/2}}\sum_{i=0}^{2^m-1}|i\rangle \ .$$

In the beginning, every process applies this gate to its main register $|Leader\rangle\,|Coin\rangle$ consisting of $3\log+1$ qubits, (line 2). This way the entire system is in the following state:

$$\left(\frac{1}{\sqrt{2}\cdot n^{3/2}}\sum_{i=0}^{2n^3-1}|i\rangle\right)^{\otimes n} = \left(\frac{1}{\sqrt{2}\cdot n^{3/2}}\sum_{i=0}^{2n^3-1}|i\rangle\right)\otimes\ldots\otimes\left(\frac{1}{\sqrt{2}\cdot n^{3/2}}\sum_{i=0}^{2n^3-1}|i\rangle\right).$$

Observe, that if all processes measure the main registers in the computational basis, they the outcome has the same probability distribution as a random experiment in which every process draws a number from uniform distribution over $[1,\ldots,n^3]$ (first $3\log n$ qubits of each register) and a uniform random bit (the last qubit of each register).

2. **CNOT** and **Pairwise_CNOT.** The $CNOT$ is a 2-qubit gate given by the following unitary transform

$$CNOT\left(\alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle\right) = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|11\rangle + \delta\,|10\rangle.$$

The **Pairwise_CNOT** gate works on a register of $2\cdot m$ qubits. Let A be the first half of the register while B is the second. The gate applies the $CNOT$ gate qubit-wisely to corresponding pairs of qubits of A and B. We only use this gate when the part B of the entire register is in the state $|0\ldots0\rangle$. If the part A of the enitre register is in a state $\sum_{i=0}^{2^m-1}\alpha_i\,|i\rangle_A$ then the unitary gate results in

$$\texttt{Pairwise\_CNOT}\left(\sum_{i=0}^{2^m-1}\alpha_i\,|i\rangle_A, |0\ldots0\rangle_B\right) = \sum_{i=0}^{2^m-1}\alpha_i\,|i\rangle_A\,|i\rangle_B \ .$$

Whenever a process decides to send its qubits to another process, it performs the **Pairwise_CNOT** gate on its qubits and the new block of $3\log n + 1$ qubits initialized to $|0\ldots0\rangle$. Since, as a result, the information needed to be propagated is now also encoded on another $3\log n + 1$ qubits, the processes can send the new block to another part keeping the original qubits for itself. Observe, however, that this is not a universal copy gate (which is not possible in quantum model of computation), since it does not copy the part A to the part B, but rather achieves a specific form of entanglement that encodes the same information if described in the computational basis.

3. **F_CNOT** and **Controlled_Swap.** Let $f : \{0,1\}^m \to \{0,1\}$ be a Boolean function. Let A be a register of m qubits and C be an additional single qubit register. The gate **F_CNOT**$_f$ performs

NOT operation on the last qubit iff $f(A) = 1$. If $C = |0\rangle$, then we get

$$\texttt{F\_CNOT}_f \left(\sum_{i=0}^{2^m-1} \alpha_i \,|i\rangle_A \,,|0\rangle_C \right) = \sum_{i=0}^{2^m-1} \alpha_i \,|i\rangle_A \,|f(i)\rangle_C \ .$$

The gate can be implemented using 2^m unitary transforms. For each vector x from the computational basis such that $f(x) = 1$, we apply *NOT* on the last qubit and then compose at most 2^m such gates. We note here that we never use the `F_CNOT` gate on registers of more than $3\log n + 1$ qubits, therefore the `F_CNOT` has polynomial size.

The `Controlled_Swap` operates on a control register C and two registers of the same size, A and B. It swaps the content of two registers of the same size if the control register is $|1\rangle$. Formally, the gate works as follows:

$$\texttt{Controlled\_Swap}(|i\rangle_A \,,|j\rangle_B \,,\alpha\,|0\rangle + \beta\,|1\rangle) = \alpha \cdot |i\rangle_A \,|j\rangle_B \,|0\rangle + \beta \cdot |j\rangle_A \,|i\rangle_B \,|1\rangle \ .$$

We use these two gates whenever a process receives qubits of another process. First, a control qubit $|S\rangle$ is prepared by comparing the content of the received register and the main register of the process. The condition function used in the gate is the following:

$$f : \{0,1\}^{3\log n+1} \times \{0,1\}^{3\log n+1} \to \{0,1\} \quad , \quad f(i,j) \iff i > j \ .$$

Next, the qubit $|S\rangle$ is passed to the `Controlled_Swap` gate, operating again on the main register and the newly received qubits. For the $|1\rangle$ part of the control qubit, which corresponds to the fact that the received register has larger values in the computational basis, the gate switches the content of the two registers.

Let us now explain how all the gates listed above work together. As mentioned in the description of the Hadamard gate, after line 2 ends, the main registers of the system are in the state being a uniform superposition of all vectors of the computational basis. Starting from this point, the composition of all gates applied to different registers along the execution can be viewed as a single unitary gate on the entire system, consisting of the qubits that any processes ever created. Note that the unitary transformation might be different depending on the failure in communication, i.e., a failure in delivery of some block of qubits between two processes may result in abandoning gates involving these qubits, but for a moment let us assume that the links are reliable. Since the unitary transformation is linear, it is enough to consider how it affects the vectors of the computational basis. However, all the gates described above behave in the computational basis as their classic equivalents. More precisely, let $|x\rangle$ be a vector from the computational basis spanning the whole circuit. Let p be the process whose main register has the largest[‖] value in the state $|x\rangle$. From the point of view of this register, the following happens in the algorithm. In each round, p creates an entangled state on $6\log n + 2$ qubits (see point (2)) that has the same qubits on its new block of $3\log n + 1$ qubits as it has on the main register. Then, it propagates the new block to its neighbors (line 6). The neighbors compare the content of received qubits and exchange them with their main register if their content is smaller (gates `F_CNOT` and `Controlled_Swap` in lines 18). This operation is then repeated $(k+2)^2(\gamma_\alpha + 1)$ on the set of links defined by some random evolving graphs, see the later paragraph about adaptive communication pattern. In the end, the processes who, either directly or via relays, received the content of the largest main register, have the same value in their main register. Therefore, the result of the measurement in line 27 must be the same in all these processes.

[‖] The probability of a tie is polynomially small

Assume now that we are able to achieve bidirectional quantum communication between any pair of processes of an α-fraction of the entire system, regardless of the (dynamic) actions of the adversary. From the above, the algorithm transforms any vector whose largest main register is one of the registers of the α-fraction to a vector such that processes from the α-fraction have the same values in main registers. Connecting the above discussion with the fact the algorithm is a $1-$to-1 transformation on the linear space of states, we get that the probability of measuring the same values in the processes of the α-fraction is at least the probability of measuring the largest value in one of the processes belonging to the α-fraction, if the measurement was done before the quantum communication. Since the state is a uniform superposition at that time, the probability is at least $\alpha - o(1)$ and we can claim the following lemma.[**]

Lemma 1. *Let A be a set of correct processes such that any pair of them was connected by quantum communication either directly or by relays. Then the probability that all processes from A output the same bit from the algorithm* CHEAPQUANTUMCONSENSUS *is at least $\frac{|A|}{n} - o(1)$.*

Adaptive communication pattern. As explained, we not only need that the communication should be efficient in terms of the number of qubits and classic bits, but also it should be such that any two correct processes of a large fraction of the entire system are connected by a short path of correct process so that quantum registers can be relayed. Let d, α be two integers parameters. We define $k = \left\lceil \frac{\log(n/d)}{\log \alpha} \right\rceil$, $\gamma_\alpha = \frac{\log n}{\log \alpha}$, and $\delta_\alpha = \frac{2}{3}\alpha$. Initially, each process p draws independently $k + 1$ sets $\mathcal{N}_p(d), \mathcal{N}_p(d\alpha^1), \ldots, \mathcal{N}_p(d\alpha^k)$, where a set $\mathcal{N}_p(d\alpha^i)$, for $0 \le i \le k$, includes each process from $\mathcal{P}$ with probability $\frac{d\alpha^i}{n}$.

Communication is structured into $(k + 2)^2$ *epochs*, see line 4. Each epoch consists of $2(\gamma_\alpha + 1)$ communication rounds, also called *testing* rounds. They are scheduled in $\gamma_\alpha + 1$ iterations within the loop "for" in line 7, each iteration containing two communication rounds (underlined in the pseudocode): sending/receiving inquiries in line 9 and sending/receiving responses in line 12. In the testing rounds of the first epoch, a process p sends inquiries to processes in set $\mathcal{N}_p(d)$. The inquired processes respond by sending in the next round (line 12) their current classic state and specially prepared, in line 6, quantum register. However, if in a result of crashes p starts receiving less than δ_α responses per round, it switches its communication neighborhood from $\mathcal{N}_p(d)$ to the next, larger set, $\mathcal{N}_p(d \cdot \alpha)$. A similar adaptation to a crash pattern is continued in the remaining epochs.

Process p stores the cardinally of the set being inquired in an epoch in the variable $\texttt{degree}_p$ (initialized to d in line 2). For the purpose of testing rounds, p copies the value $\texttt{degree}_p$ to a variable $\texttt{adaptive\_degree}_p$ (line 6). In every testing round, p adapts its variable $\texttt{adaptive\_degree}_p$ to the largest value $x \le \texttt{adaptive\_degree}_p$ such that it received at least δ_a responses from processes that have their variable $\texttt{adaptive\_degree}$ at least x (loop "while" in line 19). If p had to decrease the value $\texttt{adaptive\_degree}_p$ in testing rounds of an epoch, it then *increases* the main variable $\texttt{degree}_p$ by the factor α before the next epoch, see line 24. The latter operation formally encodes the intuition that the process p expected to have δ_α non-faulty neighbors with their values of $\texttt{degree}$ at least as big as its own, but due to crashes it did not happen; Therefore, p increases the number of inquired processes, by adopting the next, larger neighborhood set $\mathcal{N}_p(\cdot)$, randomly selected, in order to increase the chance of communication with the majority of non-faulty processes in the next epoch. On the other hand, the adaptive procedure of reducing $\texttt{adaptive\_degree}$ in testing rounds of a single epoch helps neighbors of p to estimate correctly the size of the neighborhood that process p is using in the current testing round, which might be much smaller than the value

---

[**]The $o(1)$ part contributes to the unlikely event that there are ties between largest values.

Algorithm 2: CHEAPQUANTUMCOIN for process p

---

input: p, two parameters: d, α

1 For $0 \le i \le \left\lceil \frac{\log(n/d)}{\log \alpha} \right\rceil$: $\mathcal{N}_p(d\alpha^i) \leftarrow$ a set of processes such that each process is chosen independently
   with probability $\frac{d\alpha^i}{n}$;

2 $\texttt{degree}_p \leftarrow d, \quad \gamma_\alpha \leftarrow \frac{\log n}{\log \alpha}, \quad \delta_\alpha \leftarrow \frac{2}{3}\log n$;

3 $|Leader\rangle_p |Coin\rangle_p \leftarrow H^{\otimes n} |00\ldots0\rangle$ (a gate on $3\log n + 1$) ;

4 for $i = 1$ to $(k+2)^2$;   /* iter. of epochs */

5 do

6 $\quad$ $\texttt{adaptive\_degree} \leftarrow \texttt{degree}_p$;

7 $\quad$ **for** $j = 1$ to $\gamma_\alpha + 1$;   /* iter. of testing rounds */

8 $\quad$ **do**

9 $\quad\quad$ send to each process in $\mathcal{N}_p(\texttt{degree}_p)$: an inquire bit 1 ;

10 $\quad\quad$ $\mathcal{I} \leftarrow$ the set of processes who sent an inquire bit to p ;

11 $\quad\quad$ $\forall_{q \in \mathcal{I}} : |B_q\rangle \leftarrow \texttt{Pairwise\_CNOT}\left(|LeaderCoin\rangle_p, |0\ldots0\rangle\right)$;

12 $\quad\quad$ send to each process $q \in \mathcal{I}$: a quantum message containing first $\log^3 n$ bits of $|B_q\rangle$,
   and a classical message containing $\texttt{adaptive\_degree}_p$;

13 $\quad\quad$ $\mathcal{R} \leftarrow$ the set of processes who responded to p's inquires ;

14 $\quad\quad$ **foreach** $q \in \mathcal{R}$ **do**

15 $\quad\quad\quad$ $|CLeader\rangle_q |CCoin\rangle_q \leftarrow$ received quantim bits from $q, \quad |S\rangle \leftarrow |0\rangle$;

16 $\quad\quad\quad$ $\texttt{F\_CNOT}_{Leader_p > CLeader_q}\left(|Leader\rangle_p, |Leader\rangle_q, |S\rangle\right)$;

17 $\quad\quad\quad$ $\texttt{Controlled\_Swap}\left(|LeaderCoin\rangle_p, |CLeaderCCoin\rangle_q, |S\rangle\right)$;

18 $\quad\quad$ **end**

19 $\quad\quad$ **while** $\left|\{q \in \mathcal{R} : adaptive\_degree_q \ge adaptive\_degree_p\}\right| < \delta_\alpha$ and
   $adaptive\_degree_p \ge d$;   /* adapting #neighbors during testing */

20 $\quad\quad$ **do**

21 $\quad\quad\quad$ $\texttt{adaptive\_degree}_p \leftarrow \frac{1}{\alpha}\texttt{adaptive\_degree}_p$;

22 $\quad\quad$ **end**

23 $\quad$ **end**

24 $\quad$ **if** $adaptive\_degree_p < degree_p$ **then**

25 $\quad\quad$ $\texttt{degree}_p \leftarrow \min\{\texttt{degree}_p \cdot \alpha, d\alpha^k\}$;   /* neighborhood for next epoch grows */

26 end

27 $b_p \leftarrow$ be the last bit the measurement of $|Leader\rangle_p |Coin\rangle_p$ in the computational basis;

28 return b_p ;   /* p outputs random bit */

---

degree_p from the beginning of the epoch. This, in turn, calibrates the value of `adaptive_degree` of the neighbors of p, and this calibration can propagate to other processes of distance up to γ_α from p in the next iterations of testing rounds.

Analysis. Let us define graphs $\mathcal{G}(d\alpha^i)$, for $0 \le i \le k$, as the union of random sets $\cup_{p \in \mathcal{P}} \mathcal{N}_p(d\alpha^i)$. The probability distribution of the graph $\mathcal{G}(d\alpha^i)$ is the same as the random graph $G(n, y)$ for $y = \frac{d\alpha^i}{n}$. Chlebus, Kowalski and Strojnowski [13] showed in their Theorem 2, applied for $k = \frac{64n}{d\alpha^{i-1}}$, that the graph $\mathcal{G}(d\alpha^i)$ has the following properties, whp:

(i) it is $(\frac{n}{d\alpha^{i-1}})$-expanding, which follows from $(\frac{n}{d\alpha^{i-1}}, \frac{2}{3}\frac{n}{d\alpha^{i-2}}, \frac{4}{3}\frac{n}{d\alpha^{i-2}})$-edge-expanding property,
(ii) it is $(\frac{n}{d\alpha^{i-1}}, \frac{1}{3}\alpha, \frac{2}{3}\alpha)$-edge-dense,   (iii) it is $(16\frac{n}{d\alpha^{i-1}}, 3/4, \frac{2}{3}\alpha)$-compact,
(iv) the degree of each node is at most $\frac{21}{20}d\alpha^i$.

Since the variable degree_p takes values in the set $\{d, d\alpha^1, \ldots, d\alpha^k\}$, the pigeonhole principle assures that eventually p will participate in an epoch in which degree_p has not been increased (in the most severe scenario, p will use the set $\mathcal{N}_p(d\alpha^k)$, which consists of all other processes – because it contains each process, as a neighbor of p, with probability 1). The $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood property of random graphs composed from neighborhoods $\mathcal{N}(\text{degree}_p)$ implies that p will then contact a majority of other non-faulty processes via at most $\gamma_\alpha + 1$ intermediate processes (during testing rounds). Formally, the following holds:

Lemma 2. *If a process p does not change its variable degree_p at the end of an epoch i, then at the beginning of epoch i there exists a $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood of p in the graph $\mathcal{G}(\text{degree}_p)$ consisting of non-faulty processes with variable degree being at least degree_p in the epoch i, whp.*

On the other hand, $(16n/d\alpha^{i-1}, 3/4, 2\alpha/3)$-compactness of the (random) graph composed of processes that have the variable degree at least $d\alpha^i$, guarantees that the total number of processes that use sets $\mathcal{N}(d\alpha^i)$ during the epoch i is at most $\frac{n}{\alpha^{i-2}}$, which amortizes communication complexity.

Lemma 3. *For any integer i, such that $0 \le i \le k$, at the beginning of each epoch there is at most $\frac{16n}{d\alpha^{i-2}}$ processes with the variable degree greater than $d\alpha^i$, whp.*

The above discussion yields the following result.

Theorem 3. *For two integer parameters $d, \alpha \in \Omega(\log n)$, the algorithm QUANTUMCOINFLIP generates a weak global coin, provided that at most $\frac{1}{3}$-fraction of initially non-faulty processes have crashed. It terminates in $O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$ rounds and with high probability uses $O\left(\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \log n\right)$ both quantum and classic communication bits (amortized) per process.*

6   Constant-Time Communication-Efficient Fuzzy Counting

Although generating a weak global coin in a constant number of rounds against an adaptive adversary requires quantum communication (due to the lower bound by Ben-Or and Bar-Joseph [8]), the CHEAPQUANTUMCOIN algorithm, even without quantum communication, achieves few other goals. As discussed in the previous section, its random communication pattern guarantees, whp, that any additional classic message, also called a rumor, of a non-faulty process can be conveyed to any other non-faulty process if added to every classic message sent/received in line 12. Even more, assume that there is a set of x messages/rumors $M = \{m_1, \ldots, m_x\}$, distributed as inputs among some subset of processes (one message from M per process). If processes always convey all the known rumors from set M when using classic communication (avoiding repetition), then they

solve a Gossip problem, whp, i.e., every rumor m_i given to a non-faulty process, for $1 \le i \le x$, is known at the end of the protocol to every other non-faulty process. Observe that processes resign from the quantum content of communication for this purpose, and instead of $\log n$ bits (or qubits) per message, they use $|M|$ bits, where $|M|$ denotes the number of bits needed to encode all rumors from M. Finally, if processes work in a model where the names of other processes are commonly known, they can withdraw from using random communication. Instead, they can use a deterministic family of graphs $\mathcal{G}(d\alpha^i)$, for the same choice of parameters d and α. The proof of existence of such graphs, using the probabilistic argument, was described in [13] (Theorem 2). In such case, the set $\mathcal{N}_p(d\alpha^i)$ is the neighborhood of process p in the deterministic graph $\mathcal{G}(d\alpha^i)$. (Processes compute the same copies of graphs $\mathcal{G}$ locally in the beginning of the algorithm.) The above augmentation of the algorithm, together with the proof of Theorem 3, from which we take the upper bound on the number of messages send and the upper bound on the number of rounds, leads to the following result.

Theorem 4. *For integer parameters $d, \alpha \in \Omega(\log n)$, there is a deterministic algorithm that solves the gossip problem in $O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$ rounds using $O\left(\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \cdot (|M| + \log n)\right)$ communication bits per process (amortized), where $|M|$ is the number of bits needed to encode all rumors in M.*

Generalized Fuzzy Counting.

Definition 2 (Fuzzy Counting [25]). *An algorithm solves Fuzzy Counting if each process returns a number between the initial and the final number of active processes. Here, the notion of "being active" depends on the goal of the counting, e.g., all non-faulty processes, processes with initial value 1, etc.*

Note that the returned numbers could be different across processes. Here, we refine the state-of-art method of solving the fuzzy counting problem, even deterministically, and propose a new recursive algorithm with any branching factor x, called FASTCOUNTING. Formally, we prove the following:

Theorem 5. *For any $2 \le x \le n$ and $\delta, \alpha \in \Omega(n)$, the FASTCOUNTING deterministic algorithm solves the Fuzzy Counting problem in $O\left(\frac{\log n}{\log x}\left(\frac{\log n}{\log \alpha}\right)^3\right)$ rounds, using $O\left(\frac{\log n}{\log x}\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \cdot x \log n\right)$ communication bits (amortized) per process.*

Obviously, the constant-time is achieved in Theorem 5 when $x, \alpha = n^\epsilon$, for a constant $\epsilon \in (0, 1)$. In this case, the number of rounds is $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$, while the communication complexity is $O(n^{3\epsilon} \log^2 n)$ (amortized) per process. In our approach, we generalize the method of Hajiaghayi et al. [25] to denser communication graphs of certain properties, which allows us to achieve constant running time. The constant running time is the key feature of the algorithm, since it is used in the implementation of (expected) constant-round quantum consensus protocol CHEAPQUANTUMCONSENSUS. The main difference between ours and the state-of-art approach is a different Gossip protocol used in the divide-and-conquer method.

The FASTCOUNTING algorithm is recurrent. It takes as an input the following values: $\mathcal{P}$ is the set of processes on which the algorithm is executed; p is the name of a process which executes the algorithm; $a_p \in \{0, 1\}$ denotes if p is active ($a_p = 1$) or not; and parameters x, d, α, where x is the branching factor and d, α steer the density of the communication graph in the execution. Knowing the set $\mathcal{P}$ of n processes, FASTCOUNTING splits the set into x disjoint groups of processes, each of size between $\left\lfloor \frac{n}{x} \right\rfloor$ and $\left\lceil \frac{n}{x} \right\rceil$. Name these groups $\mathcal{P}_1, \ldots, \mathcal{P}_x$. The groups are known to each participating process. The algorithm then makes x parallel recursive calls on each of these groups.

As a result, a process p from a group $\mathcal{P}_i$, for $1 \le i \le x$, gets the number of the active processes in its group $\mathcal{P}_i$. In the merging step, all processes execute Gossip algorithm, proposed in Theorem 4, with parameters d, α, where the input rumors are the numbers calculated in the recursive calls. To keep the communication small, when processes learn new rumors they always store at most one rumor corresponding to each of the x groups. This way, the number of bits needed to encode all rumors is $O(x \log n)$. Let $r_1, \dots, r_\ell$ be the rumors learned by process p from the execution of the Gossip algorithm. The final output of p is the sum $\sum_{i=1}^{\ell} r_i$. We provide the pseudocode of the algorithm in Figure 3. Note that all processes run the algorithm to help counting and Gossip, not only those with $a_p = 1$, but only the latter are counted (see lines 2, 5, 7). For detail analysis of the round and communication bit complexity we refer to Section C.

Algorithm 3: FASTCOUNTING for process p

    input: $\mathcal{P}, p, a_p; x, d, \alpha$

1   **if** $|\mathcal{P}| = 1$ **then**

2 | **return** a_p

3   $\mathcal{P}_1, \dots, \mathcal{P}_x \leftarrow$ partition of $\mathcal{P}$ into x disjoint group of size between $\left\lfloor \frac{|\mathcal{P}|}{x} \right\rfloor$ and $\left\lceil \frac{|\mathcal{P}|}{x} \right\rceil$;   `/* this`
    `partition is common for all processes, since we assumed the common knowledge of`
    `set` $\mathcal{P}$ `*/`

4   let g be the index of the p's group, i.e., $p \in \mathcal{P}_g$;

5   `active`$_p \leftarrow$ FASTCOUNTING$(\mathcal{P}_g, p, a_p, x, d, \alpha)$;

6   $\mathcal{R} \leftarrow$ the outcome of the Gossip algorithm, referred to in Theorem 4, executed on the set of processes $\mathcal{P}$ with initial message `active`$_p$ and parameters d, α;

7   **return** $\sum_{q \in \mathcal{R}} active_q$

References

[1] I. Abraham, T.-H. Hubert Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, Communication Complexity of Byzantine Agreement, Revisited, in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2019, pp. 317 - 326.

[2] D. Alistarh, J. Aspnes, V. King, and J. Saia, Communication-efficient randomized consensus, *Distributed Computing*, 31 (2018), 489 - 501.

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

[4] J. van Apeldoorn and T. de Vos, A Framework for Distributed Quantum Queries in the CONGEST Model. *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2022.

[5] J. Aspnes, Lower Bounds for Distributed Coin-Flipping and Randomized Consensus, *J. ACM* 45(3) (1998): 415 - 450.

[6] H. Attiya, and K. Censor, Tight bounds for asynchronous randomized consensus, Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007

[7] H. Attiya, and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, 2nd edition, Wiley, 2004.

[8] Z. Bar-Joseph, and M. Ben-Or, A Tight Lower Bound for Randomized Synchronous Consensus, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1998, 193 - 199.

[9] Z. Bar-Joseph, and M. Ben-Or, (2002) A Tight Lower Bound for Randomized Synchronous Consensus. DOI: 10.1145/277697.277733

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

[11] A. Broadbent and A. Tapp, Can quantum mechanics help distributed computing? *SIGACT News*, 39(3):67-76, 2008.

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

[13] B.S. Chlebus, D.R. Kowalski, and M. Strojnowski, Fast scalable deterministic consensus for crash failures, in *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC)*, 2009, pp. 111 - 120.

[14] B.S. Chlebus, D.R. Kowalski, M. Strojnowski, Scalable Quantum Consensus for Crash Failures, in *Proceedings of the 24th International Symposium on Distributed Computing (DISC)*, 2010.

[15] V. Cholvi, Quantum Byzantine Agreement for Any Number of Dishonest Parties, *Quantum Inf. Process.*, 21 (2022) 1 - 11.

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

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

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

[19] P. Feldman, and S. Micali, An optimal probabilistic protocol for synchronous Byzantine agreement, *SIAM Journal on Computing*, 26 (1997) 873 - 933.

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

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

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

[23] S. Gilbert, and D.R. Kowalski, Distributed agreement with optimal communication complexity, in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, pp. 965 - 977.

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

[25] M. Hajiaghayi, D.R. Kowalski, and J. Olkowski, Improved communication complexity of fault-tolerant consensus, in *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, 2022, to appear.

[26] L.K. Helm, Quantum distributed consensus, in *Proceedings of the 27th ACM symposium on Principles of distributed computing (PODC)*, 2008.

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

[28] Q. Luo, K. Feng, and M. Zheng, Quantum multi-valued byzantine agreement based on d-dimensional entangled states, *International Journal of Theoretical Physics*, 58 (2019) 4025 - 4032.

[29] M. A Nielsen and I. Chuang, Quantum computation and quantum information, *American Association of Physics Teachers*

[30] D. Peleg, (2000). Distributed Computing: A Locality-Sensitive Approach. SIAM.

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

[32] S. Tani, H. Kobayashi, and K. Matsumoto, Exact Quantum Algorithms for the Leader Election Problem, *ACM Transactions on Computation Theory*, 4 (2012) 1 - 24.

[33] X. Wu, and P. Yao, Quantum Complexity of Weighted Diameter and Radius in CONGEST Networks, *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2022

APPENDIX

A The Analysis of CheapQuantumConsensus Algorithm

To analyze the CHEAPQUANTUMCONSENSUS algorithm we first recall a combinations of lemmas from [9].

Lemma 4 (Lemmas $4.1, 4.2, 4.3$ in [9])**.** *If all processes have the same value at the beginning of an iteration of the main while loop, then the algorithm returns the decision after at most two iterations.*

Theorem 6. *For any $\epsilon > 0$, the CHEAPQUANTUMCONSENSUS algorithm solves Consensus against $n/3$-adversary in $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds in expectation while using $O(n^{3\epsilon})$ qubits and communication bits per process (amortized), whp.*

Proof. First, we argue for correctness. Compared to the protocol of Bar-Joseph and Ben-Or [9], which works for an arbitrary number of failures $t \leq n$, we changed the method of deriving random coin, c.f., line 18. Observe that even if CHEAPQUANTUMCOIN fails to meet conditions of t-resilient coin flip, it always outputs some bit in every non-faulty processes. Thus, regardless of number of crashes the output could be an output of local coin flips with a non-zero probability. Since Bar-Josephs's and Ben-Or's algorithm works with probability 1 (see Theorem 3 in [9]), thus CHEAPQUANTUMCONSENSUS also achieves correctness with probability 1.

Next, we estimate the expected number of phases (i.e. the number of iterations of the main while loop). We consider only *good* phases, i.e. phases in which the adversary crashed at most $\frac{1}{10}$ fraction of processes that were correct at the beginning of this iteration. Note, that there can be at most $\frac{1}{3}/\frac{1}{10} < 4$ "bad" phases. Let x be the number of non-faulty processes at the beginning of some good phase. We consider following cases:

Case *a)* There exists a process that in this iteration executes line 15 or line 14. In this case, all other processes have to execute line 14 or line 18, since the number of ones counted in different processes may differ by $\frac{x}{10}$ at most. All processes that in this iteration execute CHEAPQUANTUMCOIN will set b_p to 1 with probability $\frac{1}{4}$ at least. What follows, in the next iteration all processes will start with b_p set to 1 and by Lemma 4 the algorithms will decide within two next phases.

Case *b)* There exists a process that in this phase executes line 16 or line 17. Similarly to the previous case, we observe that all other processes have to execute line 17 or line 18, since, again, the number of ones counted in different processes may differ by $\frac{x}{10}$ at most. Therefore the same arguments applies, but know the final decision will be 0.

Case *c)* None of processes executes one of lines 14, 15, 16, or 17. Thus, all processes participated in CHEAPQUANTUMCOIN in line 18. By Theorem 3, with probability at least $\frac{1}{4}$, all processes will set value b_p to the same value. Thus, again by applying Lemma 4, we get that the algorithms will decide within two next phases.

We showed that if good phase happens, then the algorithm terminates within 2 next iterations with probability at least $\frac{1}{4}$. Since there can be at most 4 bad iterations, thus we can calculate the expected number of iterations as follows

$$\mathbb{E}(ITE) = \sum_{i=4}^{\infty} i\left(\frac{1}{4}\right)^i = O(1) \ .$$

Executing a single phase takes $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds, which is the round complexity of the FASTCOUNTING algorithm and an upper bound of the time complexity of the CHEAPQUANTUMCOIN algorithm, therefore the algorithm terminates in $O\left(\left(\frac{1}{\epsilon}\right)^4\right)$ rounds in expectation. Similarly, by taking the upper bounds on the communication complexity of the algorithms FASTCOUNTING and CHEAPQUANTUMCOIN we get that the expected number of amortized communication bits used by the algorithm is $O(n^{3\epsilon})$. $\square$

Finally, let us analyze the modified version of the algorithm CHEAPQUANTUMCONSENSUS with the difference that processes use parameters $x := 2, d = \log n, \alpha := \log n$ in lines 3 and 18.

Theorem 7. *The modified version of the* CHEAPQUANTUMCONSENSUS *algorithm solves Consensus against any adversary in* $O(\log^5 n)$ *rounds in expectation while using* $O(\log^8 n)$ *qubits and communication bits per process (amortized), whp.*

Proof. For the correcntess we argue exactly the same as in the proof of previous Theorem. We also define good and bad phases, with only this differences that now the number of bad phases is at most $O(\log n)$, since the adversary has the full power of crashing arbitrary number of processes. This being said we get that, by the very same reasoning, that the expected number of phases is

$$\mathbb{E}(ITE) = \sum_{i=\log n}^{\infty} i \left(\frac{1}{4}\right)^i = \Theta(\log n) \ .$$

By examining the time and bits complexity of the algorithms FASTCOUNTING and CHEAPQUANTUMCOIN with parameters $x = 2, d = \log n, \alpha = \log n$, we get a single phase lasts $O(\log^4 n)$ rounds and contributes $O(n \log^7 n)$ bits to the total communication complexity. The latter, after dividing by n, gives the sought complexity amortized per process. Thus, the theorem follows. □

B   Omitted proofs from Section 5

Proof of Lemma 2. Consider two last iterations of the epoch i. Since the variable $\texttt{degree}_p$ has not changed it the epoch, thus also the variable $\texttt{adaptive\_degree}_p$ has remained unchanged in the epoch. In particular, examining line 19 assures, that in the last two testing rounds of this epoch, process p received at least δ_α responses from processes that had the variable $\texttt{adaptive\_degree}$ at least $\texttt{degree}_p$.

Next, we proceed by induction. We claim, that in the $2j$ and $2j - 1$, for $1 \le j \le \gamma_\alpha$ last testing rounds there existed (j, δ_α)-dense-neighborhood of p with the properties the every processes belonging to this neighborhood is non-faulty and has its variable $\texttt{adaptive\_degree}$ at least $\texttt{adaptive\_degree}_p$. The induction step goes as follows. Assume that there exists (j, δ_α)-dense-neighborhood of p with the mentioned properties. In particular in the $2(j + 1), 2(j + 1) - 1$ last testing rounds, every process from this set had to receive at least δ_α responses from processes with the variable $\texttt{adaptive\_degree}$ at least as big as the variable $\texttt{adaptive\_degree}$ of the process. This follow from inspecting line 19. The set of processes who responded with large values of $\texttt{adaptive\_degree}$ constitute to the set $N^{j+1}(p)$.

Eventually, we obtain the existence of the set $N^{\gamma_\alpha}(p)$ which satisfies the property of $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood. Each process from the neighborhood has the variable $\texttt{degree}$ at least $\texttt{degree}_p$, since for every process q it holds that $\texttt{degree}_q \ge \texttt{adaptive\_degree}_q$. □

Proof of Lemma 3. Assume to the contrary that there exists an epoch such that more than $\frac{16n}{3\alpha^{i-2}}$ processes start this epoch with the variable $\texttt{degree}$ set to a value greater than $d\alpha^i$. Assume also w.l.o.g. that i is the *first* such epoch and let A be the set of processes that have the variable $\texttt{degree}$ set to at least $d\alpha^i$ at the beginning of this epoch. Let C be any survival set for A with respect to the graph $\mathcal{G}(d\alpha^{i-1})$. Note that since A is a set of processes that are correct in epoch i, thus the processes from C have been behaving correctly in epoch $1, \ldots, i - 1$ inclusively. Since the graph $\mathcal{G}(d\alpha^{i-1})$ is $(16\frac{n}{d\alpha^{i-2}}, 3/4, \frac{2}{3}\alpha)$-compact and $|A| \ge 16\frac{n}{d\alpha^{i-2}}$, thus C exists. At the beginning of the epoch i, the variable $\texttt{degree}$ of all processes from C is greater than $d\alpha^i$, thus there must be a round $j < i$ in which the last process from C, say q, increases its variable $\texttt{degree}$ to $d\alpha^i$. But this gives us

the contradiction. In the epoch j, all processes from C operates in the graph $\mathcal{G}(d\alpha^{i-1})$, thus they have at least δ_α neighbors in C. All these neighbors have the variable $\mathtt{degree}$ greater than $d\alpha^{i-1}$. In particular, in this epoch, the process q will not execute line 24 and therefore it will not increase its variable $\mathtt{degree}_p$ which give a contradiction with the maximality of j and in consequence the minimality of i. $\square$

Note that in the above proof of Lemma 3 we use the fact that a suitable set C of processes that have been correct throughout the whole epoch exists. We may choose this set and argue about it after the epoch, as the communication pattern in the algorithm is deterministic. Hence, in any round of the epoch, processes in C have at least as many non-faulty neighbors in communication graph as they have neighbors from set C. We use this number of neighbors in C as a *lower bound* to argue about behavior of variables $\mathtt{degree}$; therefore, our arguments do not depend on behavior of processes outside of C and whether/when some of them fail during the epoch.

Lemma 5. *Any two non-faulty processes p and q were connected by a quantum path of communication during the run of the algorithm.*

Proof. First observe, that the variable $\mathtt{degree}_p$ can take at most $k+1$ different values. Since there is $(k+2)^2$ epochs, thus by the pigeonhole principle there must be a sequence of consecutive $k+2$ epochs in which the variable $\mathtt{degree}_p$ remains unchanged. Similarly, among these $k+2$ epochs there must be two epochs in which the variable $\mathtt{degree}_q$ remains unchanged. Let us denote these two epochs i and $i+1$. Since the variable $\mathtt{degree}_p$ has not changed in the epoch $i+1$, thus by applying Lemma 2, we get that there exists a $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood of p in the graph $\mathcal{G}(\mathtt{degree}_p)$, say A, consisting of processes that a) were non-faulty in the epoch b) their variable $\mathtt{degree}$ were at least $\mathtt{degree}_p$. An immediate consequence is that all processes from A were non-faulty in the epoch i. Moreover, since A is $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood of p in the graph $\mathcal{G}(\mathtt{degree}_p)$ thus its size is at least $\frac{n\alpha}{\mathtt{degree}_p}$ (Lemma 1 [13]). Let B be the analogues $(\gamma_\alpha, \delta_\alpha)$-dense-neighborhood of q derived from the properties of the graph $\mathcal{G}(\mathtt{degree}_q)$.

Now, consider quantum communication between sets A and B in the first testing round of the epoch $i+1$. Processes from A use the graph $\mathcal{G}(\mathtt{degree}_p)$ to communicate (or a denser graph) while processes from B use the graph $\mathcal{G}(\mathtt{degree}_q)$ (or a denser graph). The graph $\mathcal{G}(\mathtt{degree}_p)$ is $(\frac{n\alpha}{\mathtt{degree}_p})$-expanding and the graph $\mathcal{G}(\mathtt{degree}_q)$ is at least $(\frac{n\alpha}{\mathtt{degree}_q})$-expanding. Therefore, at least one process from set A inquires a process from set B when executes line 9 in this testing round if $\mathtt{degree}_p \geq \mathtt{degree}_q$; or vice-versa if $\mathtt{degree}_p < \mathtt{degree}_q$. Since the set B has diameter γ_α (Lemma 1 [13]) and there remains γ_α testing rounds in the epoch $i+1$, thus by the end of this epoch any quantum messages send from p could reach q. $\square$

Proof of Theorem 3. Let $\mathcal{H} \subseteq \mathcal{P}$ be the set of initially non-faulty processes. Assume that at least $\frac{2}{3}|\mathcal{H}|$ of them remains non-faulty during the execution of the algorithm. By Lemma 5, any pair of processes from $\mathcal{H}$ is connected by quantum communication, therefore applying Lemma 1 to this set, gives that there is at least $\frac{2}{3} - o(1)$ (which is greater than $\frac{1}{2}$ for sufficiently large n) probability that all non-faulty processes return the same output bit. Since 0 and 1 are equally likely, thus the probabilistic guarantee on the weak global coin follows.

The number of rounds is deterministic and upper bounded by $O(k^2 \cdot \gamma_\alpha) = O\left(\left(\frac{\log(n/d)}{\log \alpha}\right)^2 \frac{\log n}{\log \alpha}\right) = O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$. To bound the communication complexity, assume that each graph $\mathcal{G}(d\alpha^i)$, for $0 \leq i \leq k$, satisfies the desired expanding properties listed in the description of the algorithm. This, by the union bound argument, holds whp. By Lemma 3 at the beginning of each epoch there is at most $\frac{16n}{d\alpha^{i-2}}$ processes that inquires more than $d\alpha^i$ other processes in testing rounds of this epoch,

for each $0 \leq i \leq k$. Since each message uses at most $O(\log n)$ bits and qubits, thus a single testing round in an epoch adds at most

$$\sum_{i=0}^{k} \frac{16n}{d\alpha^{i-2}} \cdot d\alpha^i \leq 16kd\alpha^2 \cdot n \log n$$

qubits and bits to communication complexity. Since there is exactly $O\left(\left(\frac{\log n}{\log \alpha}\right)^3\right)$ testing rounds, thus the $O\left(\left(\frac{\log n}{\log \alpha}\right)^4 d\alpha^2 \cdot n \log n\right)$ upper bound on the total communication complexity of the algorithm follows. Dividing the latter by n we receive amortized formula per process. $\square$

C The Analysis of FastCounting algorithm

Proof of Theorem 5. Let us first argue for the correctness. If $|\mathcal{P}| = 1$, then the conditional statement in line 1 proves the correctness. Next, we proceed by induction. Assume that the correctness is proved for any subset $\mathcal{P}' \subseteq \mathcal{P}$. Therefore, whenever a process p executes a recursive call in line 5, its variable $\texttt{active}_p$ is assigned to the number being an estimation of non-faulty active processes in the set $\mathcal{P}_g$. Consider now set $\{\texttt{active}_p : p \in \mathcal{P}\}$. This is the set of input messages to the execution of the Gossip algorithm, see line 6. Thus, by Theorem 4, we get that unless all process from a group $\mathcal{P}_p$ crashes, then all other non-faulty processes will have the number $\texttt{active}_p$ in their sets $\mathcal{R}$ (line 6). Therefore, the sum returned in line 7 is the actual estimate of the number of all-non faulty processes in the set $\mathcal{P}$, since it can differ from the number of all processes starting the algorithm only by the number of crashed processes.

The bounds on time and communication complexity follows immediately from the bounds on complexity of the Gossip algorithm proposed in Theorem 4 and the fact that depth of the recursion at most $O(\log n / \log x)$. $\square$

D Further Results

Lemma 6. *For any Consensus algorithm there is a strategy of an adaptive adversary under which some process sends $\Omega(n)$ messages with non-zero constant probability.*

Proof. Suppose otherwise. Consider any process p and two executions, in one of them p starts with value 0 and in the other starts with 1. In both executions, the adversary fails all processes that want to send a message to p and all to whom p sends a message. For each pair (p, q), we obtain two probabilities $\alpha(p, q), \beta(p, q)$, of p sending a message to q in the first and the second execution. The sum of probabilities $\sum_{q \in G} \alpha(p, q) + \beta(p, q)$ is $o(n)$ with constant probability $c > 0$, by contradictory assumption (recall that this sum is for the fixed process p).

Repeat the above for every process p. This way we obtain a complete weighted graph (by weights $\alpha(\cdot, \cdot)$ and $\beta(\cdot, \cdot)$). Again, by contradictory assumption, the total sum of all weights is $o(n^2)$ with the same constant probability $c > 0$, therefore, because the number of pairs is $n(n-1)$, there is a pair p, q such that

$$\alpha(p, q) + \alpha(q, p) + \beta(p, q) + \beta(q, p) = o(1) .$$

Consider an execution $\mathcal{E}$ in which the adversary crashes all but processes p, q in the very beginning. It assigns value 0 to p and value 1 to q. It follows that in this execution p, q try to communicate directly with probability $\alpha(p, q) + \beta(q, p) = o(1)$. With complementary probability, $1 - o(1)$, they do not communicate. In this case, p cannot distinguish this execution $\mathcal{E}$ from another execution $\mathcal{E}_p$ in

which all processes start with value 0 but the adversary crashes all of them but p in the beginning – hence, by validity condition of consensus, p must decide 0 in both of these executions. Similarly, q cannot distinguish the constructed execution $\mathcal{E}$ from execution $\mathcal{E}_q$ in which all processes start with value 1 but the adversary crashes all of them but q in the beginning – hence, by validity condition of consensus, q must decide 1 in both of these executions. Both the above facts must hold with probability 1, as validity is required with probability 1. Consequently, in the constructed execution cE, p decides 0 while q decides 1, with probability $1 - o(1)$, which violates agreement condition of consensus. This contradiction concludes the proof of the lemma. □

Next, we argue that no constant-time Consensus algorithm uses amortized polylogarithmic number of communication bits, per process.

Lemma 7. *Any quantum algorithm solving Consensus in expected time $O(1)$ requires a polynomial number of messages per process (amortized), whp.*

Proof. Consider a Consensus algorithm working in expected τ number of rounds, for some $\tau = O(1)$. By Markov inequality, the algorithm terminates by time 2τ with probability at least $1/2$, for every strategy of the (adaptive) adversary. We base on the idea in the proof of Lemma 6, but we do it for every round $r \leq 2\tau$ and for number of sent messages in a round being $r \cdot n^{r/(2\tau)}$. Using recursive argument, we argue that there is a set A_r of at least $\Omega(\frac{n}{r \cdot n^{r/(2\tau)}})$ processes (instead of just a pair of elements as in the proof of Lemma 6) that communicate with each other with probability $o(1)$, with already sending $\Omega(n^{(r-1)/(2\tau)})$ messages, amortized per process in this set. In one of the rounds, however, a constant $(1/(2\tau))$ fraction of alive processes must increase their communication peers by factor of at least $n^{1/\tau}$, because in τ expected rounds (so also with constant probability) they need to contact $\Omega(n)$ other peers in some executions – again, by the same argument as in the proof of Lemma 6 (i.e., otherwise, we could find a pair of them, assign different values, and enforce decision on them with a constant probability, violating agreement condition). In such round, the adversary allows them to send their message without crashes, thus enforcing total number of $\Omega(\frac{n}{r \cdot n^{r/(2\tau)}} \cdot n^{(r-1)/(2\tau)} \cdot n^{1/\tau}) = \Omega(n^{1+1/(2\tau)})$ messages, i.e., at least $\Omega(n^{1/(2\tau)})$ messages amortized per every process in the system. As this happens with a constant probability, the proof follows. □