Deterministic Fault-Tolerant Local Load Balancing and its Applications against Adaptive Adversaries

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

Abstract

Load balancing is among the basic primitives in distributed computing. In this paper, we consider this problem when executed locally on a network with nodes prone to failures. We show that there exist lightweight network topologies that are immune to message delivery failures incurred by (at most) a constant fraction of all nodes. More precisely, we design a novel deterministic fault-tolerant local load balancing (LLB) algorithm, which, similarly to their classical counterparts working in fault-free networks, has a relatively simple structure and guarantees exponentially fast convergence to the average value despite crash and omission failures.

As the second part of our contribution, we show three applications of the newly developed fault-tolerant local load balancing protocol. We give a randomized consensus algorithm, working against $t < n/3$ crash failures, that improves over the best-known consensus solution by Hajiaghayi et al. (STOC'22) with respect to communication complexity, yet with an arguable simpler technique of combining a randomly and locally selected virtual communication graph with a deterministic fault-tolerant local load balancing on this graph.

We also give a new solution for consensus for networks with omission failures. Our solution works against $t < \frac{n}{C \log n (\log\log n)^2}$ omissions, for some constant C, is nearly optimal in terms of time complexity, but most notably – it has communication complexity $O((t^2 + n) \text{ polylog } n)$, matching, within a polylogarithmic factor, the lower bound by Abraham et. al. (PODC'19) with respect to both terms depending on t and n. Ours is the first algorithm in the literature that is simultaneously nearly optimal, in terms of n, t, with respect to both complexity measures, against the adaptive omission-causing adversary. Finally, we show an application of load balancing to the problem of almost everywhere counting in networks prone to crash or omission failures. Our techniques improve the communication complexity of these applications, compared to the best-known algorithms, by at least a polylogarithmic factor, and in case of consensus against omission failures – even by a polynomial.

[1]School of Computer and Cyber Sciences, Augusta University, Augusta, Georgia, USA.
[2]Independent.

1 Introduction

A fault-tolerant ***local load balancing (LLB)*** is a distributed problem in which every process (also called a vertex or a node) starts with an input value in $[0, 1]$, and at the end there exists a subset of processes, of size $n - O(t)$, such that the values stored by these processes differ by at most $\tilde{O}(t/n)$ from the mean μ of all the input values, where n is the number of all processes and t is the number of faulty ones.[1]

Message-passing system. We consider a message-passing fully connected distributed system of n processes. The processes know n (or its linear estimate). They work synchronously in rounds. Each of them knows only its local numbering of communication ports, but the port numbers do not automatically provide information about the identity of the other side of the adjacent link. We also do not require the processes to have a unique ID, i.e., they could be anonymous. Messages sent via point-to-point links are of size $O(\log n)$ each, but a process may send different messages via different ports in a round (multicast operation).

Our LLB algorithm can work efficiently not only in fully-connected message-passing systems, as described above, but also in a class of well-connected network topologies (see Def. 1). The considered applications, however, require full connectivity of the communication network.

Process failures. We consider two standard types of process' failures: crashes and omissions. Suppose a *crash failure* of a process occurs in a round of computation. In that case, only some arbitrary subset of links adjacent to this process succeeds in delivering messages from the process in this round, and starting from the next round, the process halts its computation entirely: no messages can be sent by nor delivered to this process, and the process is excluded from any further consideration. An *omission failure* of a process means that its adjacent links could occasionally omit some in-coming and out-going messages, starting from the round in which this process becomes faulty.

Adversary. In any type of failures, we assume that an *adaptive full-information adversary* controls the pattern of these failures. More specifically, the adversary knows the protocol and can control chosen processes, up to t of them, together with their adjacent links, in an online fashion based on the history of the computation in the system. If processes use randomness, we assume that the adversary can see the entire system's state at any moment of the execution, but cannot foresee future random bits to be provided to the processes (if any).

Load balancing is a widely applicable primitive – we demonstrate efficient applications of our fault-tolerant LLB algorithm to the problems of counting and consensus under crash and omission failures.

Counting. In the ***k-almost-everywhere counting problem***, there is a subset of processes with a raised flag, and each process knows in the beginning only whether its flag is raised or not.
Correctness: At the end, there is a subset of processes, of size at least k, such that the counting algorithm at each process in this subset returns the number of processes with raised flag, with an additive accuracy of $\tilde{O}(t)$.

Consensus. In the ***binary consensus problem*** each process is given an input value in $\{0, 1\}$ and the goal of every correct process is to decide on some of the processes' input, adhering to the three requirements:
Validity: Only a value among the initial ones may be decided upon.
Agreement: No two non-faulty processes decide on different values.
Termination: Each process eventually decides on some value unless it is faulty.

We consider randomized solutions to the above problems, and thus, we require that all problem-specific conditions hold with high probability. We say that an event holds *with high probability (whp)* if there exists an absolute constant $C \geq 1$ such that the probability of this event is at least $1 - \frac{1}{n^C}$.

Our contributions

Our main technical contribution is a <u>deterministic fault-tolerant LLB</u> algorithm FaultTolerantLLB, which works under both <u>crash and omission failures</u>. The algorithm executed by a process v, takes as input a subset of ports and the load of a node corresponding to the process executing the algorithm, and returns a pair $x(v)$, `type`, where $x(v)$ is the final load at process v. If the conceptual communication graph/network G, formed by edges corresponding to the input ports at nodes (i.e., edge $\{v, w\}$ means one of the input ports in v

[1]Notation $\tilde{O}(\cdot)$ hides poly-logarithmic factors.

leads to w and vice versa), is $(d_{\min}, d_{\max})$-well-connected, i.e., it satisfies certain connectivity properties and each node has degree between $d_{\min}$ and $d_{\max}$ (see Definition 1), then our algorithm guarantees the following (here we present a simplified version of Theorem 1 for suitable constants $d_{\min}, d_{\max}$):

Theorem (Theorem 1 in Section 3, version for constant $d_{\min}, d_{\max}$)**.** *Let G be a $(d_{\min}, d_{\max})$-well-connected graph. The algorithm* FaultTolerantLLB *executed on the graph G under at most t crash or omission failures, achieves the following guarantees:*

(i) it terminates in $O(\tau_1 + \tau_2)$ rounds using $O((\tau_1 + \tau_2)|M|)$ communication bits per node, where $|M|$ is the size of machine word and $\tau_1, \tau_2 = O(\log n)$;

(ii) the first element of the returned pair is always between the largest and the smallest input value;

(iii) if $t < C \cdot n$, for some constant C depending on $d_{\min}, d_{\max}$, then there exists a set of nodes A, of size at least $n - \frac{3}{2}t$, such that for every $v \in A$, every returned pair $x(v)$, `type` *satisfies* `type` *= active;*

(iv) let $\varepsilon \in [0, 1]$ be such that $t < \frac{\varepsilon}{3\tau_1} \cdot C' \cdot n$, for some C' depending only on $d_{\min}, d_{\max}$, and let μ denote the mean of the input values; Then, for every node $v \in A$ it holds that

$$x(v) \in [\mu - \varepsilon, \mu + \varepsilon] \ .$$

The main technical advancement is in combining specific LLB formula in line 5, applied $\tau_1 = O(\log n)$ times in the main loop of the LLB Algorithm 1, with the fault-tolerant mechanism of fixing outliers that follows (see its description in Algorithm 2). The latter mechanism is checking in $\tau_2 = O(\log n)$ rounds whether enough number of still "non-suspected" neighbors remain active – positive check implicitly confirms that the preceding LLB procedure in τ_1-hop neighborhood was executed by a large-enough non-suspected subgraph of the original communication graph and thus the returned value is a good estimate of the mean value.

We could apply our LLB algorithm on pre-determined or random graphs (i.e., on pre-determined or randomly selected input ports), to get deterministic or randomized version of LLB, respectively. Hence, it could be applied to improve performance of both deterministic and randomized solutions to various distributed computing problems. To demonstrate its efficiency, we apply it to solve almost-everywhere counting and consensus under crash and omission failures.

Applying the FaultTolerantLLB algorithm to the problem of almost-everywhere counting yields:

Theorem (Theorem 2 in Section 4)**.** *For any number of either crash or omission failures $t \leq \frac{n}{\log n}$, there is an algorithm that solves $(n - 3t)$-almost-everywhere counting problem in $O(\log n)$ rounds using $O(n \log^3 n)$ communication bits, whp.*

The new counting algorithm can be applied to solve the consensus problem for crash failures.

Theorem (Theorem 3 in Section 5)**.** *There exists an algorithm that solves consensus against $t < n/3$ crashes in $O(\sqrt{n} \log^{3/2} n)$ rounds using $O\left(n^{3/2} \log^{5/2}(n) (\log \log n)^2\right)$ communication bits, whp.*

This result improves the communication complexity of the previous most efficient algorithm from [21], Theorem 2, by multiplicative factor $O(\log^{3/2} n)$. We note, however, that the algorithm in [21] achieves correctness with probability 1 and has round complexity lower by factor $O(\log^{1/2} n)$. Nevertheless, the technique proposed in this paper is much simpler and also more versatile – our model does not require knowing the identifiers, which by itself requires $\Omega(n^2)$ of total communication if not provided as the part of the code.

Finally, our technique of fault-tolerant LLB also provides a lightweight, simple and efficient consensus algorithm against omission failures, see the theorem below. Although our algorithm tolerates $O(\frac{n}{\log n (\log \log n)^2})$ faulty processes, the result holds against the adaptive, full-power adversary. Our algorithm is time-efficient, in the sense that it is polylogarithmically close to the lower bound $\Omega\left(\frac{t}{\sqrt{n \log n}}\right)$ by Bar-Joseph and Ben-Or [6]. Since $O\left(t^2 \log^2 n (\log \log n)^4 + (t\sqrt{n} + n) \log^3 n (\log \log n)^2\right) \subseteq O((t^2 + n) \text{ polylog } n)$, the communication complexity of our algorithm is also polylogarithmically close to the lower bound $\Omega(t^2 + n)$ for randomized algorithms, by Abraham et al. [1]. The best known time-efficient algorithmic solution, by Hajiaghayi et al. [20], requires $\Theta(n^2 \text{ polylog } n)$ communication bits, which is polynomially worse than ours for $t = O(n^{1-\epsilon})$.

Theorem (Theorem 4 in Section 6)**.** *There exists an algorithm that solves consensus against* $t < \frac{n}{C \log n (\log\log n)^2}$ *omission failures in* $O\left(\frac{t \log^2 n}{\sqrt{n}} + \log^2 n\right)$ *rounds using* $O\left(t^2 \log^2 n\,(\log\log n)^4 + (t\sqrt{n} + n)\log^3 n\,(\log\log n)^2\right)$ *communication bits, for some constant* C*, whp.*

Related work

A problem in which *all* processes calculate a function of certain parameters that are held at individual nodes is closely related to the local load balancing problem considered in this paper. For example, when all nodes calculate the average of these initial values, they are said to reach average consensus. Average consensus and, more generally, distributed function calculation, especially based on diffusion algorithms, have received a tremendous amount of attention by many communities, including the control community, multi-agent systems, signal-processing and sensor networks community, as well as theoretical distributed computing [16, 27, 28, 29, 32, 35]. Importantly, usually in this line of work, processes are assumed to be *always* correct and the emphasis is usually on: the pace of asymptotic convergence with respect to network's topology [23, 16, 29], and the issues of finite time completion [31, 34] or the quantized transmissions [23]. Reference [19] discusses several applications of distributed average consensus.

Some important derivations of the average consensus problem considered variations where processes are allowed to exhibit errors [18]. Nevertheless, such settings usually divert from the classical crash/omission/Byzantine failure model by that faulty processes can only manipulate received values, not the communication topology.

Load-balancing can be understood as a problem where parties are given a potentially uneven assignment of "work" which they must then attempt to spread evenly. [15, 4, 25] studied this problem when the underlying topology has a stochastic nature and derive asymptomatic convergence based on the distribution of edges in the topology graph. In [24], a deterministic load balancing was used as part of counting in dynamic networks, however there was an additional assumption that there is a leader in the network – such an assumption does not stand if faulty processes are considered (which is the case in our work).

We apply LLB to improve efficiency of consensus. Consensus problem under crash failures has been widely studied. The best deterministic solution uses time $O(t)$ and $O(n + t\log n)$ communication bits [10], while the corresponding lower bounds $\Omega(t)$ and $\Omega(n)$ were proved, respectively, in [17] and [3]. Randomized consensus for crash failures against a powerful adaptive adversary is less studied, and the recent result in [21] suggests that there could be a trade-off between time complexity, which is $\Omega\left(\frac{t}{\sqrt{n \log n}}\right)$ [6], and communication complexity, which for linear-time solution could be very small (see the deterministic case) but for fast solutions the best known is $O(n^{3/2}\text{polylog } n)$ [21]. The abovementioned lower bounds for time complexities naturally extend to more severe omission failures, while a stronger lower bound on communication $\Omega(t^2 + n)$ was proved by Abraham et al. [1].

Other than consensus, similar set of techniques and approaches has been applied to gossip - a problem where all processes start with an initial value and over time all correct processes must learn initial value of other correct processes (but can arbitrarily differ on knowledge of incorrect processes) [2, 5, 7, 14].

Another derivative of consensus is *counting* – a problem in which the correct parties' goal is to give an estimate of their number in finite time. [9] studied Byzantine resilient counting without assuming any approximate knowledge of the number of processes in the system. Counting in networks with weaker fault models was also studied. The work of [25] considers the case when faults are oblivious (i.e., set up before the algorithm could mitigate them). More restricted scenario, such as presence of a leader or a small number of failures, was studied in [24, 8, 9]. The counting problem under process crashes was studied in [21].

In this context, and to the best of our knowledge, ours is the first paper studying theoretically the problem of local load balancing, in the averaging sense, against an *adaptive, full-information* adversary causing processes' crashes and/or message omissions. Rather than on the stochastic nature of the problem, we focus on combinatorial properties of graphs that make the fault-tolerant averaging / load-balancing possible. We also do not assume any additional hints to correct processes regarding the state of the faulty ones.

Organization of the paper. Section 2 contains necessary background and notation from the combinatorial, spectral and random graph theory, and selected fault-tolerant properties of graphs. Section 3 presents the main result – local load balancing algorithm FaultTolerantLLB and its analysis. Applications to counting, and to consensus problems tolerating crash and omission failures, are given in Sections 4, 5 and 6, resp. Omitted technical proofs from Sections 2, 3 and 4 are provided in Sections 7, 8 and 9, resp.

2 Preliminaries

Graph notation. We introduce basic definitions from graph theory and spectral graph theory. Let $G = (V, E)$ denote an undirected graph. We use $N_G(v)$ to denote the neighbors of v in G and $deg(v)$ to indicate the degree of v. Let $W \subseteq V$ be a set of nodes in G. We say that an edge (v, w) of G is *internal for* W if v and w are both in W. We denote $E(W)$, the number of internal edges for 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. We denote $\partial(W_1)$, the set of edges between W_1 and $V \setminus W_1$. We denote by $vol(W_1)$ the sum of degrees of all vertices W_1.

Spectral graph theory. We define the adjacency matrix $A(G)$ (in general, we omit the reference (G) in the matrix notation if G is known from the context) as a $|V| \times |V|$ matrix with $A_{i,j} = 1$ if there is an edge between i and j, $i \neq j \in V$, and with $A_{i,j} = 0$ otherwise. We denote $D(G)$, the diagonal matrix with the degrees of vertices of G written on the main diagonal. For a diagonal matrix X with only positive entries, we define matrix $X^{-1/2}$ as a diagonal matrix with inverse square roots of the corresponding diagonal entries of X on its diagonal (and zeros outside the diagonal). The normalized adjacency matrix is then defined as $\mathcal{A} = D^{-1/2} A D^{-1/2}$. The normalized Laplacian matrix of the graph G is defined as $\mathcal{L} = I - \mathcal{A}$, where I is the identity matrix.

The eigenvalues of the matrix $\mathcal{L}$ are closely related to combinatorial properties of G. In fact, $\mathcal{L}$ has n real eigenvalues. We use $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ to denote the eigenvalues of $\mathcal{L}$. It is well-known that eigenvalues of the normalized Laplacian matrix are non-negative and do not exceed, c.f. Section 4.3.1 in [22].

Fault-tolerant properties of graphs. The connectivity properties of G are closely related with the magnitude of the second smallest eigenvalue of $\mathcal{L}$.

Lemma 1. *For any graph G with second eigenvalue λ_2 and any subset of nodes W,*

$$E(W) \leq \frac{1}{2} \cdot vol(W) \left(1 - \lambda_2 \cdot \left(1 - \frac{vol(W)}{vol(G)}\right)\right) .$$

For this reason, we will use the following definition of a well-connectedgraph.

Definition 1. *A graph G is called $(d_{\min}, d_{\max})$-well-connected if and only if*
(i) the degree of any vertex v of G, satisfies $deg(v) \in [d_{\min}, d_{\max}]$, and
(ii) $\lambda_2 \geq 1 - \frac{1}{10 \log \log n}$, where n is the number of vertices of G.

An example of a (d, d)-well-connected graph is an (n, d, λ)-expander, for $d = \omega_n(1), \lambda = o(\log \log d)$. Recall that an (n, d, λ)-expander is a d-regular graph of n nodes and of the second eigenvalue of the adjacency matrix equal to λ. We note that however similar in flavor, the definition of a $(d_{\min}, d_{\max})$-well-connected graph is broader, as it allows irregular graphs. The tolerance of the irregularity in the degrees of vertices will become crucial in later applications.

In the case of expander graphs, it was observed in [33], that removing certain linear fraction of nodes of an expander graph leaves a densely connected "core subgraph" in the remaining part of the graph. Using analogical reasoning, we recover this observation for the class of well-connected graphs.

Formal construction of core subgraph. Let G be a $(d_{\min}, d_{\max})$-well-connected graph, and let us consider the following scenario. Let F be a given set of vertices to be removed from G. In later applications, the set F will often coincide with processes that are flawed for the purpose of our algorithms. Consider an inductive construction of a sequence of sets $W_0, W_1, \ldots$ which starts with the set $W_0 := F$ and in every step

$i \geq 1$, W_i is formed by enlarging W_{i-1} by any vertex from G that has less than $\phi d_{\min}$ neighbors in $G \setminus W_{i-1}$. Since $W_0 \subseteq W_1 \subseteq W_2 \ldots$ and there are n vertices, the sequence of sets has a fixed point, which we denote W_k. Let G' be a graph induced by $V \setminus W_k$. We note that every vertex in G' has a degree at least $\phi d_{\min}$.

Lemma 2. *Let $G = (V, E)$ be a $(d_{\min}, d_{\max})$-well-connected graph, F be a subset of V such that $|F| < \alpha|V|$, for $\alpha \in (0, 1)$, and furthermore assume that the inequality relates α and ϕ: $\alpha < (1-\phi)\frac{40}{27}\frac{d_{\min}^2}{d_{\max}^2} - \frac{2}{9}\frac{d_{\min}}{d_{\max}}$. Then the size of the set W_k in the above construction of the core subgraph is at most $\left\lceil \frac{3}{2}|F| \right\rceil$.*

Random graphs. We use the Erdős–Rényi model of random graphs. In this model, each edge appears independently with probability p. The distribution of graphs with n vertices in the Erdős–Rényi model is denoted $G(n, p)$.

Lemma 3. *Let G be a random graph drawn from $G(n, p)$, for some parameter p. There exists a constant C such that if $p \geq C \cdot \frac{\log n (\log \log n)^2}{n-1}$, then the following holds with probability at least $1 - \frac{1}{n^2}$:*
(a) degree of every vertex of G is in the interval $\left[p \cdot (n-1)\left(1 - \frac{1}{20 \log \log n}\right), p \cdot (n-1)\left(1 + \frac{1}{20 \log \log n}\right)\right]$,
(b) $\lambda_2(G) \geq 1 - \frac{1}{10 \log \log n}$.

3 Fault-tolerant local load balancing

We present a novel deterministic fault-tolerant local load balancing algorithm, FaultTolerantLLB, which pseudocode is given in Algorithm 1. We prove that it can converge in a distributed system in which processes are prone to an arbitrary pattern of adaptive failures. Recall that n denotes the number of processes in the distributed system and $t < n/10$ [2] denotes an upper bound on the number of erroneous processes.

Algorithm 1: FaultTolerantLLB

```
   input: $(d_{\min}, d_{\max})$-well-connected graph $G$, $v$, $b_v$
1  $x_0(v) \leftarrow b_v$;
2  for $i \leftarrow 1$ to $\tau_1 \leftarrow 32 d_{\max}^2 / d_{\min}^2 \log n$ do
3      send $x_{i-1}(v)$ to every vertex in $N_G(v)$;
4      let $N_i(v)$ be the set of vertices from which $v$ received a message in the previous round;
5      $x_i(v) \leftarrow \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}(u)\right) + \frac{2d_{\max} - |N_i(v)|}{2d_{\max}} x_{i-1}(v)$;
6  $x(v)$, type $\leftarrow$ FixOutliers$(G, v, x_{\tau_1}(v))$;
7  return $x(v)$, type
```

Algorithm 2: FixOutliers

```
   input: $(d_{\min}, d_{\max})$-well-connected graph $G$, $v, x_0(v)$
1  type $\leftarrow$ active;
2  for $i \leftarrow 1$ to $\tau_2 \leftarrow \log n / \log\left(\frac{34}{15} - \frac{4d_{\min}}{3d_{\max}}\right)$ do
3      send $x_{i-1}(v)$ to every vertex in $N_{G_1}(v)$;
4      let $N_i(v)$ be the set of vertices from which $v$ received a message in the previous round;
5      if $|N_i(v)| < \frac{2}{3} d_{\min}$ then
6          type $\leftarrow$ silent;
7          $v$ stops communication until the loop finishes;
8      else
9          $x_i(v) \leftarrow \text{median}\{x_{i-1}(u) : u \in N_i(v)\}$;
10 return $(x_{\log_{14/15} n+1}(v)$, type$)$
```

The algorithm takes as an input a $(d_{\min}, d_{\max})$-well-connected graph G and the input load of a process that executes the algorithm. We identify the processes with the vertices of G. For the ease of presentation

[2] We do not aim here at optimizing the linear factor in the upper bound on the number of faulty nodes.

and analysis, we assume that the knowledge of G given to a node consists of a representation of the entire graph, the parameters $(d_{\min}, d_{\max})$, and the identifier of the vertex v executing this algorithm. However, the minimal required knowledge for v must contain only links connecting v to its neighbors (equivalent to edges adjacent to v in G) and the parameters $(d_{\min}, d_{\max})$. If the network is anonymous, the process executing the algorithm does not even need to know its identifier.

The load-balancing part of the algorithm is executed in lines 2-5. In these lines, we implement a natural counterpart of the classical load balancing process, but adjusted to the presence of faulty nodes. This part of the algorithm takes $\tau_1 = 32d_{\max}^2/d_{\min}^2 \log n$ rounds. This is expected, as it roughly corresponds to the number of iterations a classical error-free random walk would need to converge, in $\|\cdot\|_1$ norm, within $\frac{1}{n}$ radius to its limit distribution, if executed on a regularized version of graph G. The load of a vertex v after round $i \in [\tau_1]$ is stored in variable $x_i(v)$; the value $x_0(v)$ corresponds to the initial load of the node v.

3.1 Analysis of the main loop of the load balancing algorithm

We define x_i, for $0 \le i \le \tau_1$, as the vector of loads of the nodes after round i (following the above convention, for $i = 0$ the vector x_i has entries corresponding to the input values). To analyze the changes in the vectors $(x_i)_{0\le I\le\tau_1}$ over consecutive rounds, we define three alternatives, which abstract runs of the main loop of the algorithm under different treatment of loads of the faulty nodes.

Definition 2 (Ideal load balancing)**.** *The ideal load balancing corresponds to the run of the main loop in which the values of the vector x_0 are exchanged along edges of $G^{\mathcal{I}}$ but assuming that no node is erroneous, where $G^{\mathcal{I}}$ is the graph obtained from G by adding loops to every vertex such that $G^{\mathcal{I}}$ is γd-regular. Define*

$$x_0^{\mathcal{I}} := x_0, \textit{ and}$$

$$x_i^{\mathcal{I}}(v) := \left(\sum_{u \in N_{G^{\mathcal{I}}}(v)} \frac{1}{2d_{\max}} x_{i-1}^{\mathcal{I}}(u)\right) + \frac{2d_{\max} - |N_{G^{\mathcal{I}}}(v)|}{2d_{\max}} x_{i-1}^{\mathcal{I}}(v) \quad \forall \quad v \in V, 1 \le i \le \tau_1 \ .$$

Definition 3 (Load balancing skewed toward 0)**.** *This load balancing corresponds to the run of the main loop of the Algorithm 1 in which the loads of faulty nodes are always assumed to be 0. This skews the convergence towards 0. Recall that $N_i(v)$ denotes those nodes who send v a load in round i and define*

$$x_0^{0} := x_0, \textit{ and}$$

$$x_i^{0}(v) := \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}^{0}(u)\right) + \frac{1}{2} x_{i-1}^{0}(v) \quad \forall \quad v \in V, 1 \le i \le \tau_1 \ .$$

Definition 4 (Load balancing skewed toward 1)**.** *This load balancing corresponds to the run of the main loop of the algorithm 1 in which the loads of faulty nodes are always assumed to be 1. This skews the convergence towards 1. We define*

$$x_0^{1} := x_0, \textit{ and}$$

$$x_i^{1}(v) := \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}^{1}(u)\right) + \frac{1}{2} x_{i-1}^{1}(v) + \frac{d_{\min} - |N_i(v)|}{2d_{\max}} \quad \forall \quad v \in V, 1 \le i \le \tau_1 \ .$$

We start by analyzing the rate of convergence of the ideal load balancing represented by the sequence of vectors $\left(x_i^{\mathcal{I}}\right)_{0\le i\le\tau_1}$. To this end, we define $G^{\mathcal{I}}$ as the regularized version of the graph G in which every vertex is augmented by such a number of self-loop edges that its degree is $d_{\max}$. Note that adding a self-loop increments the degree of a vertex by 1. By the assumption that G is $(d_{\min}, d_{\max})$-well-connected, the definition of $G^{\mathcal{I}}$ is valid. Furthermore, we can utilize the spectral properties of a $(d_{\min}, d_{\max})$-well-connected graph to infer spectral properties of $G^{\mathcal{I}}$.

Lemma 4. *The second smallest eigenvalue of the normalized Laplacian of $G^{\mathcal{I}}$ satisfies: $\frac{1}{8}\left(\frac{d_{\min}}{d_{\max}}\right)^2 \le \lambda_2\left(G^{\mathcal{I}}\right)$.*

By connecting results from random walks theory with the previously proven spectral property of $G^{\mathcal{I}}$ we get:

Lemma 5. *Let $\mu = \frac{1}{n}\sum_{v\in G} x_0(v)$. For any $v \in G^{\mathcal{I}}$, it holds that $|x^{\mathcal{I}}_{\tau_1}(v) - \mu| \leq \frac{1}{n}$, where $\tau_1 = 32\frac{d^2_{\max}}{d^2_{\min}}\log n$.*

Next, we show that the entries of the vectors representing the ideal load balancing lie always in-between the entries of the vectors representing load balancing skewed toward 0 and 1, respectively.

Lemma 6. *For any $0 \leq i \leq \tau_1$ and $v \in G_1$, it holds that $x^0_i(v) \leq x^{\mathcal{I}}_i(v) \leq x^1_i(v)$.*

It also holds that the skewed load balancing processes are good approximations of the factual process described by the entries of the vectors $(x_i)_{0\leq i\leq\tau_1}$.

Lemma 7. *For any $0 \leq i \leq \log n$ and $v \in G$, it holds that $x^0_i(v) \leq x_i(v) \leq x^1_i(v)$.*

Next, we can reason about the entries of the vector x_{τ_1}, which corresponds to the distribution of the loads in the graph vertices after the termination of the main loop of the algorithm.

Lemma 8. *For any $\varepsilon \in [\frac{2}{n}, 1]$, there exists a set C of at least $n - \frac{2\tau_1}{\varepsilon}t$ vertices such that for any $v \in C$ it holds that $x_{\tau_1}(v) \in [\mu - \varepsilon, \mu + \varepsilon]$.*

3.2 Procedure FixOutliers and final analysis

By Lemma 8, there exists a set of vertices C whose loads after the load balancing stage converged to the mean value μ with the absolute error at most ε. Nevertheless, there is the remaining set of at most $2\tau_1 t/\varepsilon$ vertices which loads can by more than ε from μ. Next, we analyze the procedure FixOutliers, see Algorithm 2, that resolves this discrepancy by forcing every vertex to have the load within the accepted error or signaling the inability to do so by returning a special flag called `type` $\in \{\mathit{active}, \mathit{silent}\}$ in the pseudocode. We note that the communication in the procedure FixOutliers is performed on the same $(d_{\min}, d_{\max})$-well-connected graph G that was used in the load-balancing stage.

We call a vertex *active* if it does not become silent while executing the procedure FixOutliers; otherwise, it is *silent*. The set of all active vertices is denoted A, while the set of silent ones is denoted S. Using the properties of expanders, we can give a lower bound on the size of the set A.

Lemma 9. *If $t < \left(\frac{40}{81}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)n$, then A has size at least $n - \frac{3}{2}t$.*

We next proceed to establish the number of vertices whose load is within the absolute error of ε from the mean μ. We will analyze how the number of vertices with this property changes over the rounds of the algorithm FixOutliers. Among all subsets of V, define C_0 as a subset of maximum size satisfying the following conditions (a) every vertex from C_0 has at least $\frac{1}{2}(d_{\max}+1)$ neighbors in C_0 (b) every vertex in C_0 is active, (c) the load of any vertex in C_0 differs by at most ε from μ before round 1 starts. We first observe that there exists C_0 of large size.

Lemma 10. *If $t \leq \frac{\varepsilon}{3\tau_1}\left(1 - \frac{d_{\max}+1}{2d_{\min}}\right)\left(\frac{40}{27}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)n$, then the size of C_0 is at least $n - 3\left(\frac{\tau_1}{\varepsilon}+1\right)t$.*

We next proceed to analyze how the existence of the set C_0 influences the loads of other active vertices. We define $R_0 = A \setminus C_0$. We then define C_i, for $1 \leq i \leq \tau_2$, as the set of the vertices that are active and in round i received at least $\frac{1}{2}(d_{\min}+1)$ messages from vertices in the set C_{i-1}. Let R_i denote the set of active vertices at the end of round i that do not belong to C_i, i.e., $R_i = A \setminus C_i$. We show that based on the spectral properties of a well-connected graph, the size of set R_i diminishes exponentially between consecutive iterations.

Lemma 11. *If $t < \frac{2d_{\min}}{d_{\max}}\frac{\varepsilon n}{81}$, then for any $1 \leq i \leq \tau_2$ it holds that $|R_{i+1}| < \frac{14}{15}|R_i|$.*

Finally, we show that if a node belongs to C_i, for some $1 \leq i \leq \tau_2$, then its load is in $[\mu - \varepsilon, \mu + \varepsilon]$.

Lemma 12. *For any node $v \in C_{\tau_2}$ it holds that $x_{\tau_2}(v) \in [\mu - \varepsilon, \mu + \varepsilon]$.*

In the next theorem, we summarize the load balancing algorithm. Among properties discussed above, we also highlight that the load of every vertex is always between the smallest and the largest input value of any vertex, as all the arithmetic operations involve either a linear combination of input values with coefficients adding to 1 or taking a median.

Theorem 1. *Let G be a $(d_{\min}, d_{\max})$-well-connected graph. The algorithm* FaultTolerantLLB *executed on the graph G under at most t crash or omission failures achieves the following guarantees:*
(i) it terminates in $O(\tau_1 + \tau_2)$ rounds using $O(d_{\max}(\tau_1+\tau_2)|M|)$ communication bits per node, where $|M|$ is the size of machine word, $\tau_1 = 32\left(\frac{d_{\max}}{d_{\min}}\right)^2 \log n$, and $\tau_2 = \log n / \log\left(\frac{34}{15} - \frac{4d_{\min}}{3d_{\max}}\right)$;
(ii) the first element of the returned pair is always between the largest and the smallest input value;
(iii) if $t < \left(\frac{4}{81}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)n$ then there exists a set of nodes A, of size at least $n - \frac{3}{2}t$, such that for every $v \in A$, every returned pair $x(v)$, `type` satisfies `type` $=$ active;
(iv) let $\varepsilon \in [0,1]$ be such that $t < \frac{\varepsilon}{3\tau_1} n \cdot f(d_{\min}, d_{\max})$, for some function $f(\cdot,\cdot)$. Then for every node $v \in A$ it holds that

$$x(v) \in [\mu - \varepsilon, \mu + \varepsilon] \ ,$$

where μ denotes the mean of the input values and $f(d_{\min}, d_{\max}) = \left(1 - \frac{d_{\max}+1}{2d_{\min}}\right)\left(\frac{40}{27}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)$.

Proof. The algorithm's main loop runs for $\tau_1 = 32\left(\frac{d_{\max}}{d_{\min}}\right)^2 \log n$ rounds, with each round corresponding to one communication round. In each of these rounds, every node communicates with at most $d_{\max}$ other processes. Each message sent in this communication consists of at most one machine word. The procedure FixOutliers iterates through another loop for a fixed $O(\tau_2)$ number of times. Each iteration corresponds to one communication round, and again, each node in this round sends a message of at most one machine word to at most $d_{\max}$ other nodes. This proves property *(i)*.

To ensure that every value $x(v)$ for any node v remains between the smallest and largest input values, we observe that the values $(x_i(v))_{0 \le i \le \tau_1}$ in line 5 are updated as a weighted average of a subset of values from the previous round. Since the sum of the weights in this average is 1, a simple inductive argument shows that for any i, the value $x_i(v)$ remains within the range of the smallest and largest input values.

In the procedure FixOutliers, nodes use the median of the received values as the new value for the next round. Thus, a similar inductive argument shows that the procedure's output $x(v)$ must also lie between the smallest and largest input values. This completes the proof of property *(ii)*. The existence of the set A follows from Lemma 9, which proves property *(iii)*.

Finally, property *(iv)* follows from Lemma 11 and Lemma 12. Lemma 11 guarantees that after τ_2 rounds, the size of R_{τ_2} is smaller than 1, implying that $R_{\tau_2} = \emptyset$ and consequently, $C_{\tau_2} = A$. By applying Lemma 12, the property is proven. ☐

4 Application of LLB to Counting

We first show a relatively straightforward application of fault-tolerant LLB to the almost-everywhere counting problem. It works for both crash and omission failures and against the adaptive adversary. Pseudocode of our solution to the almost-everywhere counting problem is given in Algorithm 3, with instantiation of random links in Algorithm 4. The constant parameters $d_{\min}$ and γ used in the algorithm are defined as follows:

Algorithm 3: AlmostEverywhereLLBCounting

```
  input: $n$, $flag_v$;
1 $N(v) \leftarrow$ SetGraph($v$);
2 if $flag_v = true$ then $b_v \leftarrow 1$;
3 else $b_v \leftarrow 0$;
4 $\mu_v$, type$_v$ $\leftarrow$ FaultTolerantLLB($N(v), (d_{\min}, \gamma d_{\min}), v, b_v$);
5 if type$_v$ $= active$ then return $\mu_v$;
```

Algorithm 4: SetGraph
Constant C_2 is defined independently in the analysis of each algorithm using SetGraph as subroutine.

```
  input: $n$, $v$
1 let $p(n)$ be the positive solution to the equation $2x - x^2 = \frac{C_2 \log n (\log\log n)^2}{n-1}$;
2 let $N(v)$ be a random subset of $V \setminus \{v\}$ where each element is taken independently with
    probability $p(n)$;
3 send a dummy message to every $u \in N(v)$;
4 for every received message from $u$ add $u$ to $N(v)$;
5 return $N(v)$;
```

$d_{\min} = \frac{3}{4}\left(2p(n) - p^2(n)\right)$ and $\gamma = \frac{5}{4}$. Note that when executing the algorithm FaultTolerantLLB, we pass the minimal knowledge needed to evoke this algorithm: $N(v)$ corresponds to the of neighbors of a node v, and $(d_{\min}, \gamma d_{\min})$ describe the parameters of the implicit graph used by the nodes. Nodes that completed procedure FaultTolerantLLB as active, i.e., the procedure returned $(\mu_v, \mathtt{type}_v = \mathit{active})$, return the value μ_v as the result of counting. The next result immediately follows from Theorem 1:

Theorem 2. *For any number of either crash or omission failures $t \leq O(\frac{n}{\log n})$, the algorithm* AlmostEverywhereLLBCounting *solves $(n - 3t)$-almost-everywhere counting problem, with probability at least $1 - \frac{1}{n^2}$. With the same probability bound, it uses $O(\log n)$ rounds and $O(n \log^3 n)$ communication bits.*

5 Application of LLB to Consensus with Crash Failures

In this section, we use the load-balancing procedure to present a simple Consensus algorithm for the case of at most $n/3$ ***crash failures***, which recovers the efficiency of other state-of-the-art results. The result is based on the technique proposed in [6]; however, we use the fault-tolerant load-balancing procedure instead of the all-to-all vote-counting scheme of the former. The algorithm is called LLBConsensus:CrashFailures and is described in Algorithm 5, with random instantiation of local ports as in Algorithm 4. $Bin(1, 1/2)$ in the code denotes Bernoulli random distribution. The analysis of the algorithm is deferred to Section 9.

Other techniques for replacing the all-to-all vote-counting scheme have been proposed, e.g., the state-of-the-art result of [21]; however, fault-tolerant load balancing is much simpler while at the same time outperforming the complexities of the state-of-the-art solution. Specifically, it requires $O(\log n)$ times fewer rounds and achieves $O(\log^4 n/(\log\log n)^2)$ times better communication complexity, cf. Theorem 2 in [21].

Theorem 3. *With probability at least $1 - \frac{1}{n}$, algorithm* LLBConsensus:CrashFailures *solves consensus against $t < n/3$ crashes in $O(n^{1/2} \log^{3/2} n)$ rounds using $O(n^{3/2} \log^{5/2} n \cdot (\log\log n)^2)$ communication bits.*

6 Application of LLB to Consensus with Omission Failures

As the next application, we use the `FaultTolerantLLB` algorithm to solve consensus in networks with omissions. A pseudocode of the algorithm is given in Algorithm 6 with random instantiation of ports as in Algorithm 4. The algorithm tolerates $t < C\frac{n}{\log n (\log\log n)^2}$ omissions failures, for some constant $C < 1$.

The major difference between this algorithm and its counterpart for crash failures stems from the fact that the pattern of omission failures does not have to be monotonic, i.e., a link can fail to deliver a message in a round to/from a faulty node and reactive later. Thus, we categorize the processes into two sets: active and suspected. At any point of execution, each process is either active or suspected. Initially, all processes start as active, see line 1. A process becomes suspected if in the course of the `FaultTolerantLLB` procedure there exists at least one neighbor whose message was not received by this process. Adapting the code of the `FaultTolerantLLB` procedure to change the status of a process from trusted to suspected is straightforward, thus we only mark this new version of algorithm with an rather than rewriting the entire pseudocode, see line 5. Finally, the introduced labeling of processes is used in the process of drawing the graph, lines 3-4.

Algorithm 5: LLBConsensus:CrashFailures
Constant C_1 and parameters $d^*, d^i_{\min}, d^i_{\max}$ are defined in the analysis.

```
   input: $n$, $v$, $b_v$;
1  $N^*(v) \leftarrow$ SetGraph($v$)          // $G^*$ is $((d^*(1-1/\log\log n), d^*(1+1/\log\log n))$-well-connected
2  for $i \leftarrow 1$ to $C_1\sqrt{n}\log n$ do
3      $N_i(v) \leftarrow$ SetGraph($v$)          // $G_i$ is $(d^i_{\min}, d^i_{\max})$-well-connected
4      $\mu_v$, lb_status$_v \leftarrow$ FaultTolerantLLB($N_i(v), d^i_{\min}, d^i_{\max}, v, b_v$);
5      for $i \leftarrow 1$ to $40\log n + 1$ do
6          send ($\mu_v$, lb_status$_v$) to every vertex in $N^*(v)$;
7          $M_v \leftarrow$ set of messages received by $v$ in the previous round;
8          if $|M_v| < \frac{1}{5}d^*_{\min}$ then $v$ skips all iterations until line 10;
9          if exists $u$ such that ($\mu_u$, lb_status$_u$ = active) $\in M_v$ then
               $\mu_v$, lb_status$_v \leftarrow \mu_u$, lb_status$_u$;
10     if $\mu_v < 1/2 - \frac{1}{40}\sqrt{\frac{\log n}{n}}$ then $b_v \leftarrow 0$ ;
11     else if $\mu_v > 1/2 + \frac{1}{40}\sqrt{\frac{\log n}{n}}$ then $b_v \leftarrow 1$;
12     else $b_v \leftarrow Bin(1, 1/2)$;
13 if ever passed line 8 then ask random $10\log n$ processes about their variable $b$; upon receiving any
     response $b_u$ set $b_v \leftarrow b_u$;
14 else response $b_v$ to any inquiring process;
15 return $b_v$;
```

Algorithm 6: LLBConsensus:OmissionFailures
Constant C_1, C_2 and parameters $d^i_{\min}, d^i_{\max}$ are defined in the analysis.

```
   input: $n$, $v$, $b_v$;
1  type$_v \leftarrow$ active;
2  for $i \leftarrow 1$ to $2C_1 \max\left(\frac{t\log n}{\sqrt{n}}, \log n\right)$ do
3      if type$_v$ = suspected then skip until line 11;
4      $N_i(v) \leftarrow$ SetGraph($n, v$)          // $G_i$ is $(d^i_{\min}, d^i_{\max})$-well-connected
5      $\mu_v$, lb_status$_v \leftarrow$ FaultTolerantLLB($N_i(v), d^i_{\min}, d^i_{\max}, v, b_v$);
6      omitted$_v \leftarrow$ number of links that at least once failed to deliver a message during the last
         execution of FaultTolerantLLB;
7      if omitted$_v > 0$ then type$_v \leftarrow$ suspected;
8      if $\mu_v < 1/2 - \frac{1}{12}\sqrt{\frac{\log n}{n}}$ then $b_v \leftarrow 0$;
9      else if $\mu_v > 1/2 + \frac{1}{12}\sqrt{\frac{\log n}{n}}$ then $b_v \leftarrow 1$;
10     else $b_v \leftarrow Bin(1, 1/2)$;
11 if type$_v$ = suspected then inquire arbitrary $11C_2\log n(\log\log n)^2 t + 1$ processes about the correct
     value of $b$; set $b_v$ to any received response;
12 else respond to the requests of the inquiring processes;
13 return $b_v$;
```

The idea is to exclude suspected processed from the process of deciding consensus value as they have the potential of being failed. Therefore, once a processes marks itself suspected, it refrains from graph sampling and communication in the iterations of the main loop of the algorithm. This is feasible, because properties of the FaultTolerantLLB guarantee that a substantial fraction of all processes will not turn suspected. The ones that turn suspected, and by this fact are excluded from the knowledge of the decision value, perform an additional round of requesting active processes at the end of the algorithm.

Theorem 4. *For some constant C, the algorithm* LLBConsensus:OmissionsFailures *solves Consensus against $t < \frac{n}{C \log n (\log\log n)^2}$ omission failures in $O\left(\frac{t \log^2 n}{\sqrt{n}} + \log^2 n\right)$ rounds using $O(t^2 \log^2 n\,(\log\log n)^4 + (t\sqrt{n} + n)\log^3 n (\log\log n)^2)$ communication bits, with probability at least $1 - \frac{1}{n}$.*

6.1 Analysis of LLBConsensus:OmissionsFailures and proof of Theorem 4

We note that the notation used in this analysis is similar to that introduced in Section 9, as both algorithms utilize the same core component of the FaultTolerantLLB procedure. We require that C_1, C_2 are constants larger than 2^{15}.[3] For simplicity, we use k to denote the number of times the main loop of the algorithm is iterated, i.e., $k = 2C_1 \max\left(\frac{t \log n}{\sqrt{n}}\right)$. We set the constant C in the formulation of Theorem 4 to $15 \cdot C_2$, i.e., $t < \frac{n}{15 C_2 \log n (\log\log n)^2}$.

Let A_i denote the set of processes that are active at the beginning of iteration i of the main loop algorithm[4]. We define graph G_i as the graph formed by the processes in A_i, taking the union of the sets of edges $(N_i)\, v \in V$ at the moment they are drawn in line 2 of the procedure SetGraph. For clarity, we introduce the notation $d^i_{\text{min}} = \frac{C_2 |A_i| \log n (\log\log n)^2}{n-1}\left(1 - \frac{1}{20 \log\log n}\right)$ and $d^i_{\text{max}} = \frac{C_2 |Ai| \log n (\log\log n)^2}{n-1}\left(1 + \frac{1}{20 \log\log n}\right)$, which denote the expected minimum and maximum degree of G_i, as proven in the next Lemma.

We observe that G_i is a theoretical construction, and in practice, the sets $N_i(v)$ passed to the procedure FaultTolerantLLB may be subsets of the corresponding sets in G_i. However, this does not affect correctness. Any failure that prevents an edge from G_i from being included in $N_i(v)$—see lines 3-4 in SetGraph—can be assumed to occur in the procedure FaultTolerantLLB before the communication rounds of this procedure begin. This interpretation does not impact the correctness of FaultTolerantLLB, as in the case of a link failure, correct processes rely solely on the parameters d^imin, d^imax, and their own loads to make a transition—see line 5 in Figure 1. All this information is provided regardless of when a failure happened. Since we use the same procedure for sampling the graph G_i as in the case of crash failures in Section 5, we can state the following result.

Lemma 13. *[Lemma 24 in Section 9] If $|A_i| \geq \frac{n}{4}$, for $1 \leq i \leq k$, then the graph G_i is $(d^i_{\text{min}}, d^i_{\text{max}})$-well-connected with probability at least $1 - \frac{1}{n^2}$. Furthermore, $\frac{d^i_{\text{max}}}{d^i_{\text{min}}} \leq \frac{11}{10}$.*

In the following, we assume that for every $1 \leq i \leq k$, the graph G_i satisfies the properties of the above lemma. By the union bound argument, the probability of this event occurring is at least $1 - \frac{1}{\log^2 n^{3/2}}$. Next, we prove that the set of active processes maintains a sufficiently large size throughout all iterations of the main loop of the algorithm

Lemma 14. *For any $1 \leq i \leq k+1$, it holds that $|A_i| \geq n - 10 C_2 \log n (\log\log n)^2 t \geq \frac{n}{4}$. Furthermore, the set A_i consists of processes whose links have not failed from the beginning of the execution.*

Proof. We first note that the assumption $t < \frac{n}{15 C_2 \log n (\log\log n)^2}$ implies that $n - 10 C_2 \log n (\log\log n)^2 t \geq \frac{n}{4}$. For $1 \leq i \leq k$, let T_i denote the set of processes that experienced at least one omission failure during the communication rounds of iteration i of the main algorithm. That is, these are the processes that *failed* to send or receive at least one message during the communication rounds of iteration i. We observe that $\sum_{1 \leq i \leq k} |T_i| < \frac{n}{C_2 \log n (\log\log n)^2}$. We will show by induction that $|A_i| \geq n - \frac{3}{2} d^i_{\text{max}} \sum_{1 \leq j < i} |T_j|$. We begin with the base case $i = 1$. Here, the induction hypothesis holds trivially, as no communication occurred before the first iteration. Thus, $n - \frac{3}{2} d^i_{\text{max}} | \cup_{1 \leq j < 1} T_j| = n$, and obviously, no communication links could have failed.

Consider an iteration $1 < i \leq k+1$. Any communication rounds that occur in this iteration are invoked by calling either the procedure SetGraph or the procedure FaultTolerantLLB. However, as observed

[3]We note that while the bound on the constants may seem large, it is chosen mainly to simplify the analysis, in which we aimed at the best asymptotic complexities. By considering communication graphs that are $O(\log n)$ times denser, these constants can be significantly reduced.

[4]Note that, contrary to the definition of A_i in Section 5, an active process can be controlled by the adversary. Also, to simplify the notation, we interpret round $k+1$ as the end of iteration i.

before, without loss of generality, we can treat any failure that occurs in SetGraph as if it happens at the beginning of FaultTolerantLLB. Furthermore, following Lemma 13, we condition the analysis on the fact that the graph G_i is $(d^i_{\text{min}}, d^i_{\text{max}})$-well-connected, which holds since $|A_i| \geq \frac{n}{4}$ by the inductive assumption. Therefore, using Theorem 1, point *(ii)*, we conclude that the number of processes that return the variable `lb_status` set to true is at least $|A_i| - \frac{3}{2}|T_i|$. Now, for a process to fail to receive a message in the procedure FaultTolerantLLB, one of the following must be true: either the process itself is faulty, the process on the other side of the link is faulty, or the process on the other side of the link has become inactive during the execution of FaultTolerantLLB; otherwise, the message would be delivered. The number of processes that are faulty or become inactive during FaultTolerantLLB in iteration i is at most

$$\frac{3}{2}|T_i| + |T_i| \leq \frac{5}{2}|T_i|,$$

as previously observed. Since the maximum degree of G_i is upper bounded by d^i_{max}, at most

$$\frac{5}{2}|T_i| \cdot d^i_{\text{max}} \leq 10C_2 \log n (\log\log n)^2 |T_i|$$

processes can fail to receive a message during the communication rounds of iteration i. Combining this observation with the fact that processes that continuously receive all expected messages remain active (see lines 6-7 of the main algorithm), we obtain

$$|A_{i+1}| \geq |A_i| - 10C_2 \log n (\log\log n)^2 |T_i| \geq n - 10C_2 \log n (\log\log n)^2 \sum_{1 \leq j \leq i} |T_j|,$$

thus proving the lemma. □

In the next part of the analysis, we proceed with a proof that all active processes store the same value in the variable b upon the termination of the main loop of the algorithm. We note that the techniques used here are the same as those used in the analysis in Section 5. This is inherent due to the exploitation of the FaultTolerantLLB algorithm as the core component of both algorithms.

We call an iteration of the algorithm's main loop *safe* if at most $\frac{1}{C_1}\frac{\sqrt{n}}{\log n}$ processes stop being active in this iteration. Since a process is counted as inactive whenever it fails to receive an incoming message, we conclude that inactive processes encompass the set of erroneous vertices in the procedure FaultTolerantLLB. That is, any process that remains active is indistinguishable from a correct one. Thus, we can state the following.

Lemma 15. *Consider a safe iteration and let μ^* denote the average of values b of all process that are active at the beginning of the iteration. If v is active at the end of this iteration, the value of variable μ_v in lines 8-10 of the algorithm* LLBConsensus:OmissionFailures *satisfies* $\mu_v \in \left[\mu^* - \frac{1}{12}\sqrt{\frac{\log n}{n}}, \mu^* + \frac{1}{12}\sqrt{\frac{\log n}{n}}\right]$.

Proof. Consider a safe iteration i. By Lemma 13, the graph G_i is $(d^i_{\text{min}}, d^i_{\text{max}})$-well-connected. Thus, the outcome of the procedure FaultTolerantLLB invoked in line 4 of the main algorithm is captured by Theorem 1. In particular, simple calculations can verify that the assumptions of points (*iii*) and (*iv*) of the theorem are satisfied for the choice $\varepsilon = \frac{1}{12}\sqrt{\frac{1}{n}}$. This is because we are considering a safe iteration meaning that $t < \frac{1}{C_1}\frac{\sqrt{n}}{\log n} < \frac{1}{2^{15}}\frac{\sqrt{n}}{\log n}$, and also because by Lemma 13 we obtain that $\frac{d^i_{\text{max}}}{d^i_{\text{min}}} \leq \frac{11}{10}$. In consequence, we obtain the existence of a set B_i of size at least $|A_i| - \frac{3}{2C_1}\frac{\sqrt{n}}{\log n}$ of the property that every process from this set has the `lb_status` variable set to *active*. As only these processes stay active in an iteration of the main loop of the main algorithm, c.f. line 7, thus the lemma is proven. □

Establishing that safe iterations output a close enough approximation of the mean of values b at the beginning of the iteration, we can follow the reasoning path as in Section 5.

Lemma 16. *If an iteration i exists such that all active processes store the same value b at the end of the iteration, then all active processes have the same value b at the end of the next iteration.*

Proof. Consider the iteration $i+1$. Theorem 1, point *(ii)*, assures that the first value of the output pair of the procedure FaultTolerantLLB is the value b that active processes held at the end of the previous round. That is true for *every* process regardless of its lb_status. This value must be either 0 or 1, thus no random choices are made in the final part of the iteration $i+1$ and therefore the lemma follows. □

Lemma 17. *Consider two consecutive safe iterations $\mathcal{I}_1, \mathcal{I}_2$. Let A_i, for $i \in \{1,2\}$ be the set of active processes at the end of the iteration $\mathcal{I}_i$. With constant probability, all processes belonging to A_2 store the same value in the variable b at the end of the second iteration.*

Proof. We first recall a large deviation inequality.

Lemma 18 (Lemma 4.3 in [6])**.** *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 $t \le \sqrt{n}/8$*

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

We denote μ_1, μ_2 the average of of values of variables b held by active processes at the beginning of round $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. If $\mu_1 > \frac{1}{2} + \frac{1}{6}\sqrt{\frac{\log n}{n}}$, then by Lemma 15 ever active process v has $\mu_{\mathcal{I}_1}(v) > \frac{1}{2} + \frac{1}{12}\sqrt{\frac{\log n}{n}}$ at the end of iteration $\mathcal{I}_1$. Therefore, it assigns 1 as the value of the variable b. By Lemma 16, the variable b is also set to 1 at the end of the second iteration. Assume that $\mu_1 \le \frac{1}{2} + \frac{1}{6}\sqrt{\frac{\log n}{n}}$. Firstly, we note that in the iteration $\mathcal{I}_1$ no two processes can execute line 8 and line 9 at the same time, since the difference between right-hand-sides of these two inequalities is larger than $\frac{1}{12}\sqrt{\frac{\log n}{n}}$ while by Lemma 15 values $\mu_{\mathcal{I}_1}(v)$ can differ by at most $\frac{1}{12}\sqrt{\frac{\log n}{n}}$. This yields two cases.

Assume, that no process executes line 10. Thus all active processes change the variable b to a uniform random bit or 1. Since $|A_1| \ge \frac{n}{4}$, by Lemma 14, thus we can apply Lemma 18 for $t = \frac{1}{2}$, to conclude that with probability $\frac{1}{10^4}$, more than $\frac{1}{2}|A_1| + \frac{1}{4}\sqrt{n}$ processes assign 1 to their value b at the end of iteration $\mathcal{I}_1$. This yields

$$\mu_2 \ge \left(\frac{1}{2}|A_1| + \frac{1}{4}\sqrt{n}\right)/|A_2| \ge \frac{1}{2} + \frac{1}{6}\sqrt{\frac{1}{n}},$$

where the last inequality follows from the facts that $|A_2| \ge |A_1| - \frac{1}{C_1}\sqrt{n/\log n} \ge |A_1| - \frac{1}{C_1}\sqrt{n/\log n}$ and $|A_1| \ge n/4$. Since iteration $\mathcal{I}_2$ is also safe, thus by the reasoning analogical to the one presented at the beginning of the lemma, we conclude that all active processes assign 1 as the value of the variable b at the end of iteration $\mathcal{I}_2$.

In the second case, when no process executes line 10 we reason analogically, but only in this case, we use Lemma 18 for lower bounding the probability of deviating negatively from the expected value (i.e. we do the estimate for the expected number of 0's). □

This lets us finally reason that at the end of the last iteration, all correct processes return the same value with high probability.

Lemma 19. *With probability $1 - \frac{1}{n}$ at least, all correct processes return the same value b at the end of the algorithm.*

Proof. The algorithm iterates the main loop $k = 2C_1 \max\left(\frac{t\log n}{\sqrt{n}}, \log n\right)$ many times. Thus, by the pigeonhole principle, there is at least $C_1 \max\left(\frac{t\log n}{\sqrt{n}}, \log n\right)$ pairs of consecutive safe iterations. For every such pair, with probability at least $\frac{1}{10^4}$, all active processes store the same value in variable b at the end of the second iteration, as per Lemma 17. Once it happens, Lemma 16 assures that this value is stored in active processes until the main loop terminates. Now, the probability that every pair of safe iterations fails to unify the variable b across active processes is bounded by

$$\left(1-\frac{1}{10^4}\right)^{C_1\max\left(\frac{t\log n}{\sqrt{n}},\log n\right)} \le \left(\frac{1}{e}\right)^{\frac{C_1}{10^4}\max\left(\frac{t\log n}{\sqrt{n}},\log n\right)} \le \frac{1}{n^2},$$

as $C_1 \ge 2^{15}$.

Let us condition on the event that all active processes have the same value of the variable b upon termination of the main loop of the algorithm. It remains to show that in lines 11-12 the value stored by the active processes is transmitted to all other processes. As observed in Lemma 14, the set of active processes after the main loop end has a size larger or equal to $n - 10C_2\log n(\log\log n)^2 t$. This means that at least $n - 11C_2\log n(\log\log n)^2 t$ is active and non-faulty. In line 11, every correct, suspected process sends at least $11C_2\log n(\log\log n)^2 t + 1$ requests, thus by pigeonhole principle, at least one of these requests arrives at a correct, active process. This process must return the appropriate value b to the querying part. On the other hand, all active processes store the same value in variable b as proved at the beginning of this lemma. This guarantees that every correct process eventually returns the same value. □

Proof of Theorem 4. We first argue for correctness. The property that all returned values are the same with probability $1-\frac{1}{n}$ at least follows from Lemma 19. The fact that the value is among the input values from the point *(ii)* of Theorem 1. That is, if all processes receive the same input value, only this value is ever assigned to any variable b in the pseudocode and thus it also must be the decision value.

To derive the round and bit complexity, we observe that the number of iterations of the main loops is fixed to $O\left(\max\left(\frac{t\log n}{\sqrt{n}},\log n\right)\right) = O\left(\frac{t\log n}{\sqrt{n}}+\log n\right)$. By Theorem 1, every single iteration uses $O(\log n)$ rounds and $O\left(n\log^2(n)(\log\log n)^2\right)$ communication bits, as the maximum degrees of graphs G_i are universally bounded by Lemma 13. Multiplying the number of iterations by the complexity of a single iteration we get that the round complexity of this part of the algorithm is $O\left(\frac{t\log^2 n}{\sqrt{n}}+\log^2 n\right)$, while communication complexity if $O\left(t\sqrt{n}\log^3 n(\log\log n)^2 + n\log^3(n)(\log\log n)^2\right)$. The inquiring phase takes additional $O(1)$ rounds and $O\left(\log n(\log\log n)^2 t\right)$ communication bits per process. Following Lemma 14, there are at most $11C_2\log n(\log\log n)^2 t$ processes that are suspected, thus the total communication complexity of this stage is $O\left(t^2\log^2 n(\log\log n)^4\right)$. Thus the theorem follows. □

7 Proofs of technical results from Section 2

7.1 Proof of Lemma 1

In the proof, we use the following result:

Lemma 20 (Lemma 1 in [11])**.** *Denote λ_2 the second smallest eigenvalue of the normalized Laplacian matrix of a graph G and let S be a subset of vertices in G. Then*

$$\frac{|\partial(S)|}{\mathrm{vol}(S)} \ge \lambda_2\left(1-\frac{\mathrm{vol}(S)}{\mathrm{vol}(G)}\right).$$

We now prove Lemma 1. By definition, we have the following identity

$$E(S) = \frac{1}{2}\left(vol(S) - |\partial(S)|\right).$$

Plugging the result of Lemma 20 to the above equality proves the corollary.

7.2 Proof of Lemma 2

Assume to the contrary that the size of the set $W_k > \left\lceil \frac{3}{2}|F| \right\rceil$. It follows, that there must be an index i, $1 \le i < k$, such that $W_i = \left\lceil \frac{3}{2}\beta|F| \right\rceil$. Observe that when a vertex is added in the construction, it adds at least $(1-\phi)d_{\min}$ edges per step. Since exactly one vertex is added at a step, thus the graph W_i has at least

$$(1-\phi)\left(|W_i| - |F|\right) d_{\min} \ge (1-\phi)\left(\left\lceil \frac{3}{2}|F| \right\rceil - |F|\right) d_{\min} \ge \frac{1-\phi}{2} d_{\min}|F| \tag{1}$$

internal edges. On the other hand, using Lemma 1, we obtain that the number of internal edges in W_i is at most

$$E(W_i) \le \frac{1}{2} vol(W_i)\left(1 - \lambda_2\left(1 - \frac{vol(W_i)}{vol(G)}\right)\right) \le \frac{1}{2} d_{\max}|W_i|\left(1 - \lambda_2\left(1 - \frac{d_{\max}|W_i|}{d_{\min}|V|}\right)\right).$$

Using the assumption that: (a) $|W_i| = \left\lceil \frac{3}{2}|F| \right\rceil \le \alpha\frac{3}{2}|V|$, we can further upper bound the number of internal edges in W_i by

$$E(W_i) \le \frac{3}{4} d_{\max}|F|\left(1 - \lambda_2 + \lambda_2 \frac{d_{\max}}{d_{\min}}\frac{3}{2}\alpha\right). \tag{2}$$

Assuming that $n > 4$ and recalling that G is $(d_{\min}, d_{\max})$-well-connected, we can use using the following inequalities

$$\alpha < (1-\phi)\frac{40}{27}\frac{d_{\min}^2}{d_{\max}^2} - \frac{2}{9}\frac{d_{\min}}{d_{\max}} \text{ and } \lambda_2 \ge 1 - \frac{1}{10\log\log n} \ge \frac{9}{10}.$$

to upper bound the the right-hand side inequality in line 2. By simplifying we obtain eventually that

$$E(W_i) < \frac{1-\phi}{2} d_{\min}|F|,$$

which yields a contradiction with the inequality in line 1 and proves the lemma.

7.3 Proof of Lemma 3

We define the expected adjacency matrix of this distribution as $\bar{A} := \mathbb{E}_{G\sim G(n,p)}\left(A(G)\right)$. Similarly, we define $\bar{D} := \mathbb{E}_{G\sim G(n,p)}\left(D(G)\right)$. The definition of an expected normalized Laplacian matrix of $G(n,p)$ extends:

$$\bar{L} = I - \bar{D}^{-1/2}\bar{A}\bar{D}^{-1/2} .$$

The following theorem gives probability bounds on the connectivity properties of a random graph in which the edge appears independently with some fixed probability.

Theorem 5 (Theorem 2 in [12])**.** *Let G be a random graph where $\operatorname{pr}(v_i \sim v_j) = p_{ij}$, and each edge is independent of each other edge. Let A be the adjacency matrix of G. Let D be the diagonal matrix with $D_{ii} = \deg(v_i)$, and $\bar{D} = \mathrm{E}(D)$. Let δ be the minimum expected degree of G, and $L = I - D^{-1/2}AD^{-1/2}$ the (normalized) Laplacian matrix for G. Choose $\epsilon > 0$. If $\delta > k\ln n$, where k is positive and $k > 3\ln(4n/\varepsilon)/\ln n$ then with probability at least $1-\epsilon$, the eigenvalues of L and $\bar{L}$ satisfy*

$$\left|\lambda_j(L) - \lambda_j(\bar{L})\right| \le 3\sqrt{\frac{3\ln(4n/\epsilon)}{\delta}}$$

for all $1 \le j \le n$, where $\bar{L} = I - \bar{D}^{-1/2}\bar{A}\bar{D}^{-1/2}$.

Using this result we can give a probability bound on the fact that a graph drawn from $G(n,p)$ is $(p(n-1)(1-1/(20\log\log n)), p(n-1)(1+1/(20\log\log n)))$-well-connected, as stated in Corollary 3.

Consider a vertex v. The random variable $deg(v)$ follows the Binomial distribution with parameters $(n-1,p)$ and thus the standard Chernoff's bound gives that

$$\Pr\left[|deg(v) - p(n-1)| \ge \frac{p(n-1)}{20\log\log n}\right] \le \exp\left(-\frac{(\log\log n)^2}{400\cdot(2 + 1/\log\log n)}\cdot p(n-1)\right).$$

Bounding $1/\log\log n < 1$ and using that $p(n-1) > C \log n (\log\log n)^2$, we can set $C = 8 \cdot 400$, and calculate that

$$\Pr\left[|deg(v) - p(n-1)| \geq \frac{p(n-1)}{20\log\log n}\right] \leq \exp\big(-4\log n\big) = \frac{1}{n^4}.$$

Taking the union bound over all possible choices of the vertex v and using the complementary rule gives the lower bound $1 - \frac{1}{n^3}$ on the probability of the event (a) being satisfied.

To prove (b), we observe that if G is drawn from $G(n,p)$ then $\bar{D} = p(n-1) \cdot I$ and $\bar{A}$ satisfies $\bar{A}_{i,j} = p$ for every $i, j \in V$, $i \neq j$. This gives that $\bar{L} = I - \bar{D}^{-1/2}\bar{A}\bar{D}^{-1/2} = \big(1 + \frac{1}{n-1}\big)I - \frac{1}{n-1}\mathbf{1}_{|V|\times|V|}$. The matrix $\big(1+\frac{1}{n-1}\big)I$ has n eigenvalues $1+\frac{1}{n-1}$ and the matrix $\frac{1}{n-1}\mathbf{1}_{|V|\times|V|}$ has one eigenvalue 1 and $n-1$ eigenvalues 0. This yields that the matrix $\bar{L}$ has one eigenvalue $\frac{1}{n-1}$ and $n-1$ eigenvalues $1+\frac{1}{n-1}$. Therefore, using Theorem 5 with the parameter ϵ set to $\frac{1}{2n^2}$, $k = 12$, we get that with probability at least $1 - \frac{1}{2n^2}$

$$\lambda_2(G) \geq 1 + \frac{1}{n-1} - 3\sqrt{\frac{3\ln(4n \cdot 2n^2)}{p(n-1)}} \geq 1 - 1/(10\log\log n),$$

where the last equality holds based on the assumption that $p \geq p(n-1) > 8 \cdot 400 \log n (\log\log n)^2$. Taking the union bound for the events (a) and (b) completes the proof of Lemma 3.

8 Proofs of technical results from Section 3

8.1 Proof of Lemma 4

We start by providing a definition of edge conductance and restating Cheeger's inequality, following [26].

Definition 5 (Edge Conductance). *Let $G = (V, E)$ be an undirected graph. The conductance of a subset $S \subseteq V$ and the conductance of the graph G are defined as*

$$\phi(S) := \frac{|\delta(S)|}{\mathrm{vol}(S)} \quad \text{and} \quad \phi(G) := \min_{S:\mathrm{vol}(S)\leq|E|} \phi(S).$$

Theorem 6 (Cheeger's Inequality, Section 3 in [13]). *Let $G = (V, E)$ be an undirected graph and let λ_2 be the second smallest eigenvalue of its normalized Laplacian matrix. Then*

$$\frac{1}{2}\lambda_2 \leq \phi(G) \leq \sqrt{2\lambda_2}.$$

Using Cheeger's inequality for graph G, we get that

$$\lambda_2(G) \leq \phi(G) \leq \sqrt{2\lambda_2(G)}.$$

Observe that by the construction of $G^{\mathcal{I}}$ the degree of any vertex in $G^{\mathcal{I}}$ is at most $\frac{d_{\max}}{d_{\min}} \geq 1$ times larger than its degree in G. Thus, by the definition of conductance, we obtain that

$$\phi(G)\frac{d_{\min}}{d_{\max}} \leq \phi(G^{\mathcal{I}}).$$

On the other hand, applying Cheeger's inequality to $G^{\mathcal{I}}$ yields

$$\phi(G^{\mathcal{I}}) \leq \sqrt{2\lambda_2(G^{\mathcal{I}})}.$$

Combining the above with the previous inequalities, we get that

$$\lambda_2\left(G^{\mathcal{I}}\right) \geq \phi\left(G^{\mathcal{I}}\right)^2/2 \geq \left(\phi(G)\frac{d_{\min}}{d_{\max}}\right)^2/2 \geq \left(\lambda_2(G)\frac{d_{\min}}{d_{\max}}\right)^2/2.$$

By the fact that G is $(d_{\min}, d_{\max})$-well-connected, we get that $\lambda_2(G) = 1 - 1/(10\log\log n)$. Therefore, we can conclude that

$$\lambda_2(G^{\mathcal{I}}) \geq \left(1 - \frac{1}{10\log\log n}\right)^2\left(\frac{d_{\min}}{d_{\max}}\right)^2\frac{1}{2} \geq \left(\frac{d_{\min}}{d_{\max}}\right)^2\frac{1}{8},$$

provided that $n > 4$.

8.2 Proof of Lemma 5

We prove the property by relating the load-balancing process to a random walk on $G^{\mathcal{I}}$. A random walk on an undirected graph is a random process that chooses a random vertex according to a distribution p_0 in and in each consecutive step moves to a neighbor of the current vertex chosen from the uniform distribution of the neighboring vertices (self-loops are counted with multiplicity 1). A random walk is called *lazy* if the processes choose to sample a neighbor only with probability $\frac{1}{2}$ while with the remaining probability $\frac{1}{2}$ it stays in the current vertex. A random walk on a graph and its normalized adjacency matrix are naturally related as the latter is a compact way of encoding the probabilities of transitions in a step of the random walk. The stationary distribution is the distribution corresponding to the 1 eigenvector of the normalized adjacency matrix. In the case of a regular graph, the stationary distribution is given by the vector $\pi = \frac{1}{|V|}\mathbf{1}_{|V|}$. Random walks have been studied extensively, in particular, the expected behavior of a random walk and a graph is described by the following theorem.

Theorem 7 (Theorem 10.4.1, Section 10.5 in [30]). *Fix a graph G. Let λ_2 denotes the second smallest eigenvalue of its normalized Laplacian matrix. Denote p_t the distribution of the position of of a random walk if executed on G for t steps. For all $a, b \in V(G)$ and t, if $p_0 = \delta_a$, then*

$$|p_t(b) - \pi(b)| \le \sqrt{\frac{deg_G(b)}{deg_G(a)}}\,(1 - \lambda_2/2)^t ,$$

where δ_a denote a distribution assigning mass 1 to the vertex a.

As for the use of the theorem in our case, denote $x = \sum_v x_0^{\mathcal{I}}(v)$ and observe that $\frac{1}{x}x_0^{\mathcal{I}}$ is a probability distribution. Thus by linearity of matrix multiplication, and by Theorem 7, we obtain that, for any $v \in V$ $|\frac{1}{x}x_t^{\mathcal{I}}(v) - \frac{1}{n}| < (1 - \lambda_2/2)^t$. Using the bound from Lemma 4 and setting $t = \tau_1 = 32\,(d_{\max}/d_{\min})^2$ yields

$$\left|\frac{1}{x}x_{\tau_1}^{\mathcal{I}}(v) - \frac{1}{n}\right| < \left(1 - \frac{1}{16}\left(\frac{d_{\min}}{d_{\max}}\right)^2\right)^{\tau_1} \le \frac{1}{n^2} ,$$

where the last inequality follows from the $(1 + x)^r \le e^{xr}$ inequality. Multiplying both sides by $x < n$ finally gives

$$\left|x_{\tau_1}^{\mathcal{I}}(v) - \frac{x}{n}\right| = \left|x_t^{\mathcal{I}}(v) - \mu\right| \le x\frac{1}{n^2} \le \frac{1}{n} .$$

8.3 Proof of Lemma 6

We give a proof by induction. For the base case, we observe that $x_0^0 = x_0^{\mathcal{I}} = x_0^1$ which supports the claim.

Assume that $i \ge 1$ and that the inductive hypothesis holds. The ideal load of a vertex $v \in V$ in step i is defined as

$$x_i^{\mathcal{I}}(v) = \left(\sum_{u \in N_{G^{\mathcal{I}}}(v)} \frac{1}{2d_{\max}} x_{i-1}^{\mathcal{I}}(u)\right) + \frac{2d_{\max} - |N_{G^{\mathcal{I}}}(v)|}{2d_{\max}} x_{i-1}^{\mathcal{I}}(v) .$$

Observe that $|N_{G^{\mathcal{I}}}(v)| = d_{\max}$ and thus $\frac{2d_{\max} - |N_{G^{\mathcal{I}}}(v)|}{2d_{\max}} = \frac{1}{2}$. On the other hand, it always holds that $N_i(v) \subseteq N_{G^{\mathcal{I}}}(v)$. Using this observation, the inductive hypothesis, and the observation that the loads are always non-negative, we get that

$$\left(\sum_{u \in N_{G^{\mathcal{I}}}(v)} \frac{1}{2d_{\max}} x_{i-1}^{\mathcal{I}}(u)\right) + \frac{2d_{\max} - |N_{G^{\mathcal{I}}}(v)|}{2d_{\max}} x_{i-1}^{\mathcal{I}}(v) \ge \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}^0(u)\right) + \frac{1}{2}x_{i-1}^0(v) = x_i^0(v) .$$

This proves the first inequality of the lemma. For the second, we proceed similarly. We have that

$$\left(\sum_{u \in N_{G^{\mathcal{I}}}(v)} \frac{1}{2d_{\max}} x_{i-1}^{\mathcal{I}}(u)\right) + \frac{2d_{\max} - |N_{G^{\mathcal{I}}}(v)|}{2d_{\max}} x_{i-1}^{\mathcal{I}}(v)$$

$$\leq \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x^1_{i-1}(u) \right) + \frac{1}{2} x^1_{i-1}(v) + \frac{d - |C_i(v)|}{2d_{\max}} \; = \; x^1_i(v) \; ,$$

where the inequality follows from the inductive assumption, the observation that $\frac{2d_{\max} - |N_{G^{\mathcal{I}}}(v)|}{2d_{\max}} = \frac{1}{2}$, and the observation that the loads are always at most 1 because they are an affine combination of values that are at most 1. Thus, the lemma is proven.

8.4 Proof of Lemma 7

We use induction. For the base case, we observe that $x^0_0 = x^{\mathcal{I}}_0 = x^1_0$ which supports the claim.

Assume that $i \geq 1$ and that the inductive hypothesis holds. The real load of a vertex $v \in V$ in step i is defined as

$$x_i(v) = \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}(u) \right) + \frac{2d_{\max} - |N_i(v)|}{2d_{\max}} x_{i-1}(v).$$

We observe that $N_i(v) \subseteq N_{G^{\mathcal{I}}}(v)$ and thus $2|N_i(v)| \leq 2d_{\max}$. Therefore, $\frac{2d_{\max} - |N_i(v)|}{2d_{\max}} \geq \frac{1}{2}$. Combining this observation with the inductive hypothesis and the fact that loads are always non-negative we get that

$$\left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}(u) \right) + \frac{2d_{\max} - |N_i(v)|}{2d_{\max}} x_{i-1}(v) \geq \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x^0_{i-1}(u) \right) + \frac{1}{2} x^0_{i-1}(v) = x^0_i(v).$$

This proves the left-hand side inequality of the lemma. We use similar reasoning to get the right-hand side inequality. The inductive assumption, the observation that $|N_i(v)| \leq d_{\max}$, and the fact that all loads are always at most 1, as an affine combination of values that are at most 1, yields that

$$\left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x_{i-1}(u) \right) + \frac{2d_{\max} - |N_i(v)|}{2d_{\max}} x_{i-1}(v)$$

$$\leq \left(\sum_{u \in N_i(v)} \frac{1}{2d_{\max}} x^1_{i-1}(u) \right) + \frac{1}{2} x^1_{i-1}(v) + \frac{d - |N_i(v)|}{2d_{\max}} = x^1_i(v).$$

Thus Lemma 7 is proven.

8.5 Proof of Lemma 8

Let s^1_i (respectively s^0_i) denotes the sum of entries of the vector v^1_i (respectively v^0_i) for $0 \leq i \leq \tau_1$. Denote $s^{\mathcal{I}} = \sum_{v \in G} v^{\mathcal{I}}_0(v)$. Observe that $s^1_0 = s^0_0 = s^{\mathcal{I}}$. In every round, every edge of the graph G conveys a load of value at most $\frac{1}{2d_{\max}}$. Since there are at most t erroneous vertices, thus the number of edges that are incident to these vertices is at most $d_{\max} t$ (we accounted for the loops of every vertex), and in consequence, the absolute difference between s^1_i and s^1_{i-1} can be $t/2$ at most. It follows that $|s^1_{\tau_1} - s^{\mathcal{I}}| < \tau_1 \cdot t/2$ and, analogously, $|s^0_{\tau_1} - s^{\mathcal{I}}| < \tau_1 \cdot t/2$. By Lemma 6, $x^1_{\tau_1}$ is larger than $x^{\mathcal{I}}_{\tau_1}$ on every coordinate and thus we have

$$s^1_{\tau_1} < s^{\mathcal{I}} + \tau_1 \cdot t/2. \tag{3}$$

By the same lemma, $x^0_{\tau_1}$ is smaller than $x^{\mathcal{I}}_{\tau_1}$ on every coordinate which yields

$$s^0_{\tau_1} > s^{\mathcal{I}} - \tau_1 \cdot t/2.$$

Let C'_1 denote the set of these vertices for which $x^1_{\tau_1}(v) > s^{\mathcal{I}}_{\tau_1}(n) + \varepsilon/2$. Since for every v it holds that $x^1_{\tau_1}(v) \geq x^{\mathcal{I}}_{\tau_1}(v)$, thus to equality (3) to hold it must be that

$$|C'_1| \cdot \varepsilon/2 \leq \tau_1 \cdot t/2 \implies |C'_1| \leq \tau_1 \cdot t/\varepsilon.$$

Let $C_1 = V \setminus C_1'$. It follows that $C_1 \geq n - \tau_1 t/\varepsilon$

$$\forall_{v \in C_1} x^1_{\tau_1}(v) \leq x^{\mathcal{I}}_{\tau_1} n(v) + \varepsilon/2 \leq \mu + \varepsilon/2 + \frac{1}{n} \geq \mu + \varepsilon,$$

where the last inequality follows from Lemma 5 and from the assumption that $\varepsilon \geq \frac{1}{n}$.

Applying the same reasoning to the vector $x^0_{\tau_1}$, we obtain a set C_0 of at least $n - \tau_1 t/\varepsilon$ with the property that

$$\forall_{v \in C_0} x^0_{\tau_1}(v) \geq x^{\mathcal{I}}_{\tau_1}(v) - \varepsilon/2 \geq \mu - \varepsilon.$$

Let $C = C_0 \cap C_1$. We have that $|C| \geq n - 2\tau_1 t/\varepsilon$. Using Lemma 7, combined with the above inequalities, we get that

$$\forall_{v \in C} \quad \mu - \varepsilon \leq x_{\tau_1}(v) \leq \mu + \varepsilon,$$

which proves the lemma.

8.6 Proof of Lemma 9

Let F be the set of the erroneous vertices. By abusing the notation slightly, denote A the remaining graph obtained by applying Lemma 2 on the graph G to the set F for constants $\phi = \frac{2}{3}$ and $\alpha = \left(\frac{40}{81}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)$ (shortly, we will prove that such defined A is indeed a subset set of active vertices). Since G is $(d_{\min}, d_{\max})$-well-connected, and since $|F| = t < \left(\frac{40}{81}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right) n$, the assumptions of Lemma 2 are satisfied and we can conclude that size of A is at least $n - \frac{3}{2}|F| \geq n - \frac{3}{2}t$. Furthermore, every vertex $v \in A$ has at least $\frac{2}{3}d_{\min}$ neighbors in the subgraph induced by the vertices of A. Since A contains only correct vertices, all these $\frac{2}{3}d_{\min}$ neighbors can communicate between each other in all communication rounds of the protocol FixOutliers. Therefore, the line 6 is never executed in any vertex of A, which proves that all these vertices must remain *active*.

8.7 Proof of Lemma 10

Let C be the set of these vertices that at the beginning of the algorithm FixOutliers have the load in the interval $[\mu - \varepsilon, \mu + \varepsilon]$. Using Lemma 8, we get that $|C| \geq n - \frac{2\tau_1}{\varepsilon}t$. As the first step of the proof, we define the set $R_0 = V \setminus (A \cap C)$ as the set of these vertices that at the beginning of the algorithm FixOutliers cannot belong to C_0. Then, we apply Lemma 2 for $\phi = \left(\frac{1}{2}d_{\max} + 1\right)/d_{\min}$ to set R_0 to obtain a core set C_0^* that also satisfies all the required properties of C_0. To apply Lemma 2 we first bound the size of R_0

$$|R_0| \leq |V| - |A| + |V| - |C| \leq \frac{3}{2}t + \frac{3}{2}\left(\frac{2\tau_1}{\varepsilon} + 1\right)t \leq \left(\frac{2\tau_1}{\varepsilon} + 2\right)t \ ,$$

where the first inequality follows from the previous lower bound on the size of C and Lemma 9 that lower bounds the size of A. To satisfy the assumptions of Lemma 2 it must hold that

$$|R_0| \leq (1 - \phi)\left(\frac{40}{27}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)|V| \ ,$$

which we verify first using the previously noticed upper bound on the size of R_0 by the assumption of this lemma we have that

$$|R_0| \leq \left(\frac{2\tau_1}{\varepsilon} + 2\right)t \ ,$$

and using the lemma assumption limiting the magnitude of t

$$\left(\frac{2\tau_1}{\varepsilon} + 2\right)t \leq \left(\frac{2\tau_1}{\varepsilon} + 2\right) \cdot \frac{\varepsilon}{3\tau_1}\left(1 - \frac{d_{\max} + 1}{2d_{\min}}\right)\left(\frac{40}{27}\left(\frac{d_{\min}}{d_{\max}}\right)^2 - \frac{2}{9}\frac{d_{\min}}{d_{\max}}\right)$$

$$\leq \left(1-\left(\frac{1}{2}d_{\max}+1\right)\frac{1}{d_{\min}}\right)\left(\frac{40}{27}\left(\frac{d_{\min}}{d_{\max}}\right)^2-\frac{2}{9}\frac{d_{\min}}{d_{\max}}\right) .$$

Therefore, we get the existence of a set C_0 of size at least $n-\frac{3}{2}|R_0| \geq n-3\left(\frac{\tau_1}{\varepsilon}+1\right)t$ with the following properties (a) every vertex in C_0 has degree at least $\frac{1}{2}d_{\max}+1$ in C_0; (b) every vertex in C_0 is active since $V-R_0 \subseteq A$; (c) every vertex in C_0 has the load in the interval $[\mu-\varepsilon, \mu+\varepsilon]$ as it holds that $V-R_0 \subseteq C$.

8.8 Proof of Lemma 11

We first prove that $R_{i+1} \subseteq R_i$ for any $0 \leq i \leq \log_{14/15} n+1$. To this end, we observe that $C_i \subseteq C_{i+1}$. Indeed, by definition, every vertex belonging to C_i, for any $0 \leq i \leq \log_{14/15} n+1$, must be active. Therefore, if process v receives at least $\frac{1}{2}(d_{\max}+1)$ messages from C_{i-1} in a round i, it all receives messages from these set of neighbors in any round $j > i$ and thus it belongs to C_j for any $j \geq i$. Since $R_i = A \setminus C_i$, thus the inclusion $R_{i+1} \subseteq R_i$ follows.

By Lemma 1 the number of internal edges of R_i is at most

$$E(R_i) \leq \frac{1}{2}vol(R_i)\left(1-\lambda_2\left(1-\frac{vol(R_i)}{vol(V)}\right)\right).$$

Using the the fact that G is $(d_{\min}, d_{\max})$-well-connected , we can upper bound $vol(R_i)$ and lower bound λ_2 with $\lambda_2 \geq \frac{9}{10}$ yielding

$$\frac{1}{2}vol(R_i)\left(1-\lambda_2\left(1-\frac{vol(R_i)}{vol(V)}\right)\right) \leq \frac{1}{2}d_{\max}|R_i|\left(1-\frac{9}{10}\left(1-\frac{d_{\max}|R_i|}{d_{\min}n}\right)\right).$$

Using the observation that $R_{i+1} \subseteq R_i$, and Lemma 10 we derive that $|R_i| \leq |R_0| = |V|-|C_0| \leq 3\left(\frac{\tau_1}{\varepsilon}+1\right)t$ and thus it follows

$$\frac{1}{2}d_{\max}|R_i|\left(1-\frac{9}{10}\left(1-\frac{d_{\max}|R_i|}{d_{\min}n}\right)\right) \leq \frac{1}{2}d_{\max}|R_i|\left(\frac{1}{10}+\frac{9}{10}\frac{d_{\max}}{d_{\min}}3\left(3\frac{\tau_1}{\varepsilon}+1\right)\frac{t}{n}\right)$$

$$= \frac{1}{20}d_{\max}|R_i| + \frac{27}{20}\frac{d_{\max}^2}{d_{\min}}\left(\frac{\tau_1}{\varepsilon}+1\right)\frac{t}{n}|R_i|. \qquad (4)$$

Observe that the assumption that $t < \frac{2d_{\min}}{d_{\max}} \cdot \frac{\varepsilon n}{81}$ implies that $27\frac{d_{\max}}{d_{\min}}\left(\frac{\tau_1}{\varepsilon}+1\right)t < \frac{n}{3}$. By plugging this inequality to the left hand side of 4, we get finally, that

$$E(R_i) \leq \frac{1}{20}d_{\max}|R_i| + \frac{27}{20}\frac{d_{\max}^2}{d_{\min}}\left(\frac{\tau_1}{\varepsilon}+1\right)\frac{t}{n}|R_i| \leq \frac{1}{15}d_{\max}|R_i|.$$

On the other hand, we argue that the number of edges by which messages are sent to vertices from R_i in round $i+1$ is at least $\frac{2}{3}d_{\min}$. That follows from the inspection of line 5 of the procedure FixOutliers. Since $R_i \subseteq A$, thus in round $i+1$ every vertex maintains its active status and thus receives at least $\frac{2}{3}d_{\min}$ messages. Thus, the number of edges between C_i and R_i that transmit a message in round $i+1$ is at least

$$\left(\frac{2}{3}d_{\min}-\frac{2}{15}d_{\max}\right)|R_i.$$

Using the averaging argument, we conclude that there must be at least.

$$2\left(\frac{2d_{\min}}{3d_{\max}}-\frac{2}{15}-\frac{1}{2}\right)|R_i| = \left(\frac{4d_{\min}}{3d_{\max}}-\frac{19}{15}\right)|R_i|.$$

vertices of R_i who receive $\frac{1}{2}d_{\max}+1$ or more messages from C_i in round $i+1$ via edges that transmit messages in round $i+1$. By definition of the set C_{i+1}, it follows that all these vertices belong to C_{i+1} and thus we have that

$$|R_{i+1}| \leq \left(1-\frac{4d_{\min}}{3d_{\max}}+\frac{19}{15}\right)|R_i| = \left(\frac{34}{15}-\frac{4d_{\min}}{3d_{\max}}\right)|R_i|,$$

which proves the lemma.

8.9 Proof of Lemma 12

We reason by induction. Specifically, we prove that for any i in the range $0 \leq i \leq \tau_2$ and any vertex $v \in C_i$ it holds that $x_i(v) \in [\mu - \varepsilon, m + \varepsilon]$. The base case $i = 0$ follows immediately from Lemma 10.

For $i \geq 1$, observe that any vertex $v \in C_i$ receives at least $\frac{1}{2}d_{\max} + 1$ messages from vertices in C_i, by definition. Then the load of v in round i is set to the median of all received values in these messages. First, the maximum degree of any vertex in C_i is at most $d_{\max}$. Second, at least $\frac{1}{2}d_{\max} + 1$ received values is in the interval $[\mu - \varepsilon, m + \varepsilon]$ by the inductive assumption on C_i. Henceforth, the median of the received values must be also in $[\mu - \varepsilon, m + \varepsilon]$ and thus the lemma follows.

9 Analysis of Algorithm LLBConsensus:CrashFailures and Proof of Theorem 3

We first explain the notation used in the algorithm. The constants C_1, C_2 must be greater than 2^{155} We assume that n is the only parameter given to the algorithm, and all other expressions are either absolute constants or derived based on n. We say that a process remains *active* in execution of the algorithm if it is non-faulty and never passes the conditional statement in line 8. The remaining processes are called *silent*. Analogously, we say that a process remains active until iteration i of the main loop of the algorithm if it neither crashes nor passes the conditional statement in rounds $1, \ldots, i$.

We begin the analysis by noting the properties of the graph formed by the union of edges belonging to the sets $\{N^*\}_{v \in V}$.

Lemma 21. *Let $G^* = \cup_{v \in V} N^*(v)$. Then with probability at least $1 - \frac{1}{n^2}$ graph G^* is $(d^*(1-(1/20\log\log n)),$ $d^*(1+1/(20\log\log n)))$ well-connected, where $d^* = C_2 \log n\,(\log\log n)^2$. Also the ratio of the maximum vertex degree to the minimum vertex degree in G^* is at most $\frac{11}{10}$.*

Proof. Let us note that graph G^* is drawn from a distribution $G(n, 2p(n) - p(n)^2) = G\left(n, \frac{C_2 \log n (\log\log n)^2}{n-1}\right)$ as per the definition of $p(n)$ in line 2 of Algorithm 4. Thus, by Lemma 3 we get that G^* is $(d^*(1 - (1/20\log\log n)), d^*(1 + 1/(20\log\log n)))$-well-connected with probability at least $1 - \frac{1}{n^2}$ as we defined $d^* = C^* \log n\,(\log\log n)^2$. This give the first part of the lemma. For proving the second part, the upper bound, we note

$$\frac{d^*(1 + 1/(20\log\log n))}{d^*(1 - 1/(20\log\log n))} \leq 1 + \frac{2/(20\log\log n)}{1 - 1/(20\log\log n)}$$
$$\leq 1 + \frac{1}{10\log\log n} \leq \frac{11}{10},$$

where for the last inequality we used the fact that $n \geq 3$. □

We define $d^*_{\min} = d^*(1 - 1/(20\log\log n))$ and $d^*_{\max} = d^*(1 + 1/(20\log\log n))$. Also, in the remainder, we condition on the fact that G^* is as described in the above lemma. Next, we observe that at least $\frac{1}{4}n$ processes is active.

Lemma 22. *If $t < \frac{n}{3}$, then at in any execution at least $\frac{n}{4}$ processes remains active, whp.*

Proof. By Lemma 21, graph G^* must be $(d^*_{\min}, d^*_{\max})$ well-connected and $\frac{d^*_{\max}}{d^*_{\min}} \leq \frac{11}{10}$.

Let F denote the set of processes that crash in the execution of the algorithm. Denote $\alpha = \frac{1}{3}$ and $\phi = \frac{1}{5}$. Following the assumptions of Lemma 2, we verify that

$$(1 - \phi)\frac{40}{27}\left(\frac{d^*_{\min}}{d^*_{\max}}\right)^2 - \frac{2}{9}\frac{d^*_{\min}}{d^*_{\max}} \geq \frac{4}{5}\left(\frac{10}{11}\right)^2 - \frac{2}{9} \geq \alpha = \frac{1}{3}.$$

[5]Similarly as in Section 6, we made no effort to optimize this constant. In particular, increasing the degree of the random graph by $\log n$ decreases the requirement on the magnitude of C_1, C_2 by an order of magnitude ($\sim 2^8$), albeit at the cost of higher asymptotic complexities.

Thus by applying Lemma 2 to set F on graph G^*, with the choice of parameters $\alpha = \frac{1}{3}$, $\phi = \frac{1}{5}$, we obtain the existence of a set of processes C of size $n - \frac{3}{2}|F| \geq \frac{1}{4}n$, disjoint with F with the property, that every vertex in C has at least $\frac{1}{5}d^*_{\min}$ neighbors in C. As only correct processes constitute to the set C, we observe that none of these processes can pass the conditional statement of line 8 in the main algorithm thus proving that all these processes are active. $\square$

In the next lemma, we proof that the set of active processes can exchange any information by taking part in the communication rounds of the inner loop of an iteration of the main loop of the algorithm, e.g. lines 5-9. We say that an active process p *reaches* an active process q if there exists a path of reliable links that could transmit a message from p to q during communication in the mentioned lines during execution of these lines.

Lemma 23. *If processes p and q remain active in an iteration of the main loop of the algorithm, then p reaches q.*

Proof. Consider the process q. For $1 \leq 40 \log n$, we denote $R^i(q)$ the set of active processes who can reach q in i communication rounds of the inner loop in an iteration of the main loop of the algorithm. By induction, we will show that $|R^i(q)| \min\left((1 + 1/20)^i, n/20\right)$. The base case follows from the fact that q never executes line 8, and thus in any round of the inner loop receives $\frac{1}{5}d^*_{\min} > 1 + 1/20$ messages at least.

We now proceed to the inductive step. Consider the set $R^i(q)$. If $|R^i(q)| \geq n/20$, then the case is proven. Assume then that $|R^i(q)| < n/20$. Since all processes in $R^i(q)$ are active, thus they must receive at least $\frac{1}{5}d^*_{\min}$ messages from other active processes in every iteration of the inner loop. The number of links delivering this messages is at least $\frac{1}{5}d^*_{\min}|R^i(q)|$. On the other hand, by Lemma 1 and the fact that G^* is $(d^*_{\min}, d^*_{\max})$-well-connected, we observe that the number of edges internal to $R^i(v)$ can be bounded as follows.

$$E(R^i(v)) \leq \frac{1}{2} \cdot vol(W) \left(1 - \lambda_2 \cdot \left(1 - \frac{vol(W)}{vol(G)}\right)\right) \leq \frac{1}{2}d^*_{\min}|R^i(v)| \left(1 - \frac{9}{10}\left(1 - \frac{10}{11}\frac{1}{20}\right)\right) \leq \frac{1}{14}d^*_{\min}|R^i(v)|.$$

Therefore, at least $\frac{1}{5}d^*_{\min}|R^i(q)| - \frac{2}{14}d^*_{\min}|R^i(v)| \geq \frac{2}{35}d^*_{\min}|R^i(q)|$ of the aforementioned links connect $R^i(v)$ with $G^* \setminus R^i(v)$. Given that the maximum degree of G^* is $d^*_{\max} \leq \frac{11}{10}d^*_{\min}$, we conclude that these links are incident to at least $\frac{10}{11} \cdot \frac{2}{35}|R^i(v)|$ processes that do not belong to $R^i(v)$. It yields that $R^{i+1}(v) \geq (1 + \frac{1}{20})R^(i)(v)$ and thus the inductive hypothesis is proven. Since the inner loop iterates fro $40 \log n + 1$ steps, thus we get that $|R^{40 \log n}(v)| \geq \frac{n}{20}$.

Observe, that the same reasoning applies to p yielding another set of at least $\frac{n}{20}$ active processes that can reach p. Now, using the Expander Mixing Lemma (c.f. Lemma 2.5 in [22]), we can deduce that there is at least one edge between processes that can reach p and those that can reach q, since G^* satisfies that $\lambda_2 \geq 1 - 1/(10 \log \log n)$. Since both these sets are formed by active vertices, thus this edge serves as a reliable link between the corresponding sets in all iterations of the inner loop, ultimately proving that p reaches q. $\square$

Establishing that a constant fraction of processes remain active and information can be transmitted quickly between active processes via lines 5-9, we proceed to analyzing how the variables b change in the active processes during an execution. Let A_i be the set of these processes that are active at the beginning of iteration i of the main loop algorithm. We define graph G_i as the graph formed on the processes in A_i by taking the union of the sets of edges $(N_i)_{v \in V}$ at the moment they are drawn in line 2 of the procedure SetGraph. For clarity, we introduce the notation $d^i_{\min} = \frac{C^*|A_i| \log n (\log \log n)^2}{n-1}\left(1 - \frac{1}{20 \log \log n}\right)$ and $d^i_{\max} = \frac{C^*|A_i| \log n (\log \log n)^2}{n-1}\left(1 + \frac{1}{20 \log \log n}\right)$ that denotes the expected minimum and maximum degree of G_i as proven below.

Lemma 24. *For any $1 \leq i \leq C_1\sqrt{n \log n}$, the graph G_i is $(d^i_{\min}, d^i_{\max})$-well-connectedwith probability $1 - \frac{1}{n^2}$ at least. Furthermore, $\frac{d^i_{\max}}{d^i_{\min}} \leq \frac{11}{10}$.*

Proof. By Lemma 22, $|A_i| \geq \frac{n}{4}$. Observe that G_i is drawn from the distribution $G\left(|A_i|, 2p(n) - p(n)^2\right) = G\left(|A_i|, \frac{C_2 \log n (\log\log n)^2}{n-1}\right)$. Since C_2 is more than four times large than the equivalent constant in Lemma 3, and $4|A_i| \geq n$, thus we can apply Lemma 3 and conclude that G_i is $(d^i_{\min}, d^i_{\max})$-well-connected with probability at least $1 - \frac{1}{n^2}$. The derivation of the upper bound on the ratio of the maximum to the minimum degree of G_i follows exactly the same argument as provided in the proof of Lemma 21. □

In the remainder, we assume all graphs G_i drawn in the algorithm are $(d^i_{\min}, d^i_{\max})$-well-connected. By Lemma 24 and by the union bound argument, probability of this event is at least $1 - \frac{\log^2(n)}{n^{3/2}}$.

We call an iteration of the algorithm's main loop *safe* if at most $\frac{1}{C_1}\sqrt{n/\log n}$ processes stop being active in this iteration. Observe that a crashed processor cannot be active, thus this bound implies that at most $\frac{1}{C_1}\sqrt{n/\log n}$ crashes happens in this iteration. In our proof, we follow the argument that in a sequence of safe iterations, the load-balancing process invoked in line 4 returns an accurate enough proportion of 1's to the total number of vertices that suffices for all processes to assign the same value to the variable b with a constant probability.

Lemma 25. *Consider a safe iteration and let μ^* denote the average of values b of all process that are active at the beginning of the iteration. If v is active at the end of this iteration, the value of variable μ_v in lines 10-12 of the algorithm* LLBConsensus:CrashFailures *satisfies $\mu_v \in [\mu^* - \frac{1}{40}\sqrt{\frac{\log n}{n}}, \mu^* + \frac{1}{40}\sqrt{\frac{\log n}{n}}]$.*

Proof. Consider a safe iteration i. By Lemma 24, the graph G_i is $(d^i_{\min}, d^i_{\max})$-well-connected. Thus, the outcome of the procedure FaultTolerantLLB invoked in line 4 of the main algorithm is captured by Theorem 1. In particular, simple calculations can verify that the assumptions of points *(iii)* and *(iv)* of the theorem are satisfied for the choice $\varepsilon = \frac{1}{40}\sqrt{\frac{\log n}{n}}$. This is because we are considering a safe iteration meaning that $t < \frac{1}{C_1}\sqrt{n/\log n} < \frac{1}{2^{15}}\sqrt{n/\log n}$, and also because by Lemma 24 we obtain that $\frac{d^i_{\max}}{d^i_{\min}} \leq \frac{11}{10}$. In consequence, we obtain the existence of a set B_i of size at least $|A_i| - \frac{3}{2C_1}\sqrt{n/\log n}$ of the property that every processes from this set has the `lb_status` variable set to *active*. Reiterating the assumption that at most $\frac{1}{C_1}\sqrt{n/\log n}$ processes turn silent in this iteration, we conclude that at least $|A_i| - \frac{5}{2C_1}\sqrt{n/\log n}$ processes from the set B_i remains active until at the end of iteration i, denoted D_i. As $|A_i| \geq \frac{n}{4}$, by Lemma 22, thus $|A_i| - \frac{5}{2C_1}\sqrt{n/\log n} > 1$ and therefore D_i is not empty. Applying Lemma 23 to the set D_i and the remaining set of active processes, we conclude that any active process receives the value μ_v of an active process that terminated the FaultTolerantLLB algorithm with the `lb_status` variable set to *active*. By Theorem 1 it is irrelevant which processes value is acquired as all of them lie in the interval $\left[\mu^* - \frac{1}{40}\sqrt{\frac{\log n}{n}}, \mu^* + \frac{1}{40}\sqrt{\frac{\log n}{n}}\right]$ and the lemma is proven. □

The remaining part of the proof shows that if the accuracy in the process of counting the fraction of values b equal 1 to all active processes is precise enough, then there exists a lower bound on the probability that all active processes converge to the same value b due to the randomness invoked in the line 12. We remark that this technique has been well-established, c.f. [6, 20] in the context of synchronous consensus algorithm and we do not claim this part as the main contribution of our paper. We include the proofs for the sake of self-completeness of the paper.

We start by noting a simple fact that once a system converged to the same value of variable b stored by all active processes, this state is maintained until the end of the main loop of the algorithm.

Lemma 26. *If an iteration i exists such that all active processes store the same value b at the end of the iteration, then all active processes have the same value b at the end of the next iteration.*

Proof. Consider the iteration $i + 1$. Theorem 1, point *(ii)*, assures that the first value of the output pair of the procedure FaultTolerantLLB is the value b that active processes held at the end of previous round. That is true for *every* process regardless of its lb_status. Thus in lines 5-9 only this value circulates among active processes. This value must be either 0 or 1, thus no random choices are made in the final part of the iteration $i+1$ and thus the lemma follows. $\square$

Lemma 27. *Consider two consecutive safe iterations $\mathcal{I}_1, \mathcal{I}_2$. Let A_i, for $i \in \{1,2\}$ be the set of active processes at the end of the iteration $\mathcal{I}_i$. With probability $\sqrt{\frac{\log n}{4n}}$ all processes belonging to A_2 store the same value in the variable b at the end of second iteration.*

Proof. We first recall a large deviation inequality.

Lemma 28 (Lemma 4.3 in [6])**.** *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 denote μ_1, μ_2 the average of of values of variables b held by active processes at the beginning of round $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. If $\mu_1 > \frac{1}{2} + \frac{1}{20}\sqrt{\frac{\log n}{n}}$, then by Lemma 25 ever active process v has $\mu_{\mathcal{I}_1}(v) > \frac{1}{2} + \frac{1}{40}\sqrt{\frac{\log n}{n}}$ at the end of iteration $\mathcal{I}_1$. Therefore, it assigns 1 as the value of the variable b. By Lemma 26, the variable b is also set to 1 at the end of the second iteration. Assume that $\mu_1 \leq \frac{1}{2} + \frac{1}{20}\sqrt{\frac{\log n}{n}}$. Firstly, we note that in the iteration $\mathcal{I}_1$ no two processes can execute line 10 and line 11 at the same time, since the difference between right-hand-sides of these two inequalities is larger than $\frac{1}{20}\sqrt{\frac{\log n}{n}}$ while by Lemma 25 values $\mu_{\mathcal{I}_1}(v)$ can differ by at most $\frac{1}{20}\sqrt{\frac{\log n}{n}}$. This yields two cases.

Assume, that no process executes line 10. Thus all active processes either change the variable b to an uniform random bit or 1. Since $|A_1| \geq \frac{n}{4}$, by Lemma 22, thus we can apply Lemma 28 for $t = \frac{1}{8}\log n$, to conclude that with probability $\frac{1}{16}\sqrt{\frac{\log n}{n}}$, more than $\frac{1}{2}|A_1| + \frac{1}{16}\sqrt{n \log n}$ processes assign 1 to their value b at the end of iteration $\mathcal{I}_1$. This yields

$$\mu_2 \geq \left(\frac{1}{2}|A_1| + \frac{1}{16}\sqrt{n\log n}\right)/|A_2| \geq \frac{1}{2} + \frac{1}{20}\sqrt{\frac{\log n}{n}},$$

where the last inequality follows from the facts that $|A_2| \geq |A_1| - \frac{1}{C_1}\sqrt{n/\log n} \geq |A_1| - \frac{1}{C_1}\sqrt{n/\log n}$ and $|A_1| \geq n/4$. Since iteration $\mathcal{I}_2$ is also safe, thus by the reasoning analogical to the one presented in the beginning of the lemma, we conclude that all active processes assign 1 as the value of the variable b at the end of iteration $\mathcal{I}_2$.

In the second case, when no process executes line 10 we reason analogically, but only in this case we use Lemma 28 for lower bounding the probability of deviating negatively from the expected value (i.e. we do the estimate for the expected number of 0's). $\square$

Finally, we can prove the main theorem justifying correctness of the algorithm. In short, we use the pigeonhole principle two guarantee a large number of pairs of consecutive safe iterations. Since each pair has lower bounded probability

Lemma 29. *With probability at least $1 - \frac{1}{n}$, all correct processes return the same value b at the end of the algorithm.*

Proof. Recall, that the adversary can crash at most $n/3$ vertices. The algorithm iterates the main loop $C_1\sqrt{n\log n}$ many times. Thus, the pigeonhole principle, there is at least $\frac{C_1}{3}\sqrt{n \log n}$ pairs of consecutive safe iterations. For every such pair, with probability at least $\sqrt{\frac{\log n}{4n}}$, all active processes store the same value in variable b at the end of the second iteration, as per Lemma 27. Once it happens, Lemma 26 assures that this value is stored in active processes until the main loop terminates. Now, the probability that every pair fails to unify the variable b across active processes is bounded by

$$\left(1-\sqrt{\frac{\log n}{4n}}\right)^{C_1\sqrt{n\log n}} \le \left(\frac{1}{e}\right)^{C_1\sqrt{n\log n}\cdot\sqrt{\frac{\log n}{4n}}} = \left(\frac{1}{e}\right)^{C_1/16\log n} \le \frac{1}{2n^2} ,$$

as $C_1 \ge 2^{15}$.

Let us condition on the event that all active processes have the same value of the variable b upon termination of the main loop of the algorithm. It remains to show that in lines 13-14 the value stored by active process is transmitted to all other processes. Consider a correct correct process v. It submits $10\log n$ random requests. Since the number of active processes is at least $\frac{n}{4}$, thus Chernoff's bound implies that with probability at least $1-\frac{1}{2n^2}$ at least one of its requests hits an active process. As active processes are also non-faulty, it is guaranteed that the inquiring process receives a response with the variable b of the active process. Finally, the union bound argument provides that with probability at least $1-\frac{1}{n}$ all non-faulty processes get a response from an active processes, thus the lemma is proven. □

Proof of Theorem 3. We first argue for correctness. The property that all returned values are the same follows from Lemma 29. The fact that the value is among the input values from the point *(ii)* of Theorem 1. That is, if all processes receive the same input value, only this value is ever assigned to any variable b in the pseudocode and thus it also must be the decision value.

To derive the round and bit complexity, we observe that the number of iterations of the main loops is fixed to $O(\sqrt{n\log n})$ and, by Theorem 1, every single iteration uses $O(\log n)$ rounds and $O(n\log^2 n)$ communication bits. Finally, the inquiring phase takes additional $O(1)$ rounds and $O(n\log n)$ communication bits. □

10 Conclusions and Open Problems

We developed an efficient implementation of deterministic LLB against adaptive crashes and omissions, by using a specific LLB formula together with fixing outliers. We demonstrated substantial improvements when applying our algorithm, initiated by random links, to solve selected problems of counting and consensus (the resulting algorithms are randomized, due to the random links' initialization of the LLB sub-routine).

The most promising open direction include further applications of LLB to other distributed computing problems, as well as an attempt to extend this technique to other types of failures, in particular, different types of Byzantine faults. Further shrinking the (poly)logarithmic gaps in formulas for the number of tolerated failures, and for the time and communication complexities, is a natural challenge.

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] Dan Alistarh, Seth Gilbert, Rachid Guerraoui, and Morteza Zadimoghaddam. How efficient can gossip be?(on the cost of resilient information exchange). In *Automata, Languages and Programming: 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II 37*, pages 115–126. Springer, 2010.

[3] Eugene S. Amdur, Samuel M. Weber, and Vassos Hadzilacos. On the message complexity of binary byzantine agreement under crash failures. *Distributed Comput.*, 5(4):175–186, 1992.

[4] Natalia Amelina, Alexander Fradkov, Yuming Jiang, and Dimitrios J. Vergados. Approximate consensus in stochastic networks with application to load balancing. *IEEE Transactions on Information Theory*, 61(4):1739–1752, 2015.

[5] Tuncer Can Aysal, Mehmet Ercan Yildiz, Anand D Sarwate, and Anna Scaglione. Broadcast gossip algorithms for consensus. *IEEE Transactions on Signal processing*, 57(7):2748–2761, 2009.

[6] 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.

[7] Kai Cai and Hideaki Ishii. Quantized consensus and averaging on gossip digraphs. *IEEE Transactions on Automatic Control*, 56(9):2087–2100, 2011.

[8] Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. The hidden graph model: Communication locality and optimal resiliency with adaptive faults. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, page 153–162, New York, NY, USA, 2015. Association for Computing Machinery.

[9] Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. Byzantine-resilient counting in networks. In *42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10-13, 2022*, pages 12–22. IEEE, 2022.

[10] Bogdan S. Chlebus, Dariusz R. Kowalski, and Jan Olkowski. Deterministic fault-tolerant distributed computing in linear time and communication. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 344–354. ACM, 2023.

[11] Fan Chung. A generalized alon-boppana bound and weak ramanujan graphs. *the electronic journal of combinatorics*, 23(3):P3–4, 2016.

[12] Fan Chung and Mary Radcliffe. On the spectra of general random graphs. *the electronic journal of combinatorics*, pages P215–P215, 2011.

[13] Fan RK Chung. Laplacians of graphs and cheeger's inequalities. *Combinatorics, Paul Erdos is Eighty*, 2(157-172):13–2, 1996.

[14] Alexandros G Dimakis, Soummya Kar, José MF Moura, Michael G Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 2010.

[15] Michael Dinitz, Jeremy T. Fineman, Seth Gilbert, and Calvin Newport. Load balancing with bounded convergence in dynamic networks. In *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*, pages 1–9. IEEE, 2017.

[16] Tomaso Erseghe, Davide Zennaro, Emiliano Dall'Anese, and Lorenzo Vangelista. Fast consensus by the alternating direction multipliers method. *IEEE Transactions on Signal Processing*, 59(11):5523–5537, 2011.

[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] Christoforos N Hadjicostis and Alejandro D Dominguez-Garcia. Trustworthy distributed average consensus. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 7403–7408. IEEE, 2022.

[19] Christoforos N Hadjicostis, Alejandro D Domínguez-García, Themistokis Charalambous, et al. Distributed averaging and balancing in network systems: with applications to coordination and control. *Foundations and Trends® in Systems and Control*, 5(2-3):99–292, 2018.

[20] Mohammad Hajiaghayi, Dariusz R. Kowalski, and Jan Olkowski. Nearly-optimal consensus tolerating adaptive omissions: Why a lot of randomness is needed? In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, pages 321–331, 2024.

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

[22] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

[23] Akshay Kashyap, Tamer Başar, and Rayadurgam Srikant. Quantized consensus. *Automatica*, 43(7):1192–1203, 2007.

[24] Dariusz R. Kowalski and Miguel A. Mosteiro. Polynomial counting in anonymous dynamic networks with applications to anonymous dynamic algebraic computations. *J. ACM*, 67(2):11:1–11:17, 2020.

[25] Dariusz R. Kowalski and Miguel A. Mosteiro. Supervised average consensus in anonymous dynamic networks. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 307–317. ACM, 2021.

[26] Lap Chi Lau. Lecture notes by lap chi lau, chapters 3, 4, and 6.

[27] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49(9):1520–1533, 2004.

[28] Alessandro Pilloni, Alessandro Pisano, Yury Orlov, and Elio Usai. Consensus-based control for a network of diffusion pdes with boundary local interaction. *IEEE Transactions on Automatic control*, 61(9):2708–2713, 2015.

[29] Stefania Sardellitti, Massimiliano Giona, and Sergio Barbarossa. Fast distributed average consensus algorithms based on advection-diffusion processes. *IEEE Transactions on Signal Processing*, 58(2):826–842, 2009.

[30] Daniel Spielman. Spectral and algebraic graph theory. *Yale University*, 2025.

[31] Shreyas Sundaram and Christoforos N Hadjicostis. Finite-time distributed consensus in graphs with time-invariant topologies. In *2007 American Control Conference*, pages 711–716. IEEE, 2007.

[32] Sheng-Yuan Tu and Ali H Sayed. Diffusion strategies outperform consensus strategies for distributed estimation over adaptive networks. *IEEE Transactions on Signal Processing*, 60(12):6217–6234, 2012.

[33] 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.

[34] Long Wang and Feng Xiao. Finite-time consensus problems for networks of dynamic agents. *IEEE Transactions on Automatic Control*, 55(4):950–955, 2010.

[35] Yuqian Yang, Qingwen Qi, Jingyao Hu, Jiashu Dai, and Chengdong Yang. Adaptive fault-tolerant control for consensus of nonlinear fractional-order multi-agent systems with diffusion. *Fractal and Fractional*, 7(10):760, 2023.