Optimal Computation in Leaderless and Multi-Leader Disconnected Anonymous Dynamic Networks

Giuseppe A. Di Luna* Giovanni Viglietta†

Abstract

We give a simple characterization of which functions can be computed deterministically by anonymous processes in dynamic networks, depending on the number of leaders in the network. In addition, we provide efficient distributed algorithms for computing all such functions assuming minimal or no knowledge about the network. Each of our algorithms comes in two versions: one that terminates with the correct output and a faster one that stabilizes on the correct output without explicit termination. Notably, these are the first deterministic algorithms whose running times scale *linearly* with both the number of processes and a parameter of the network which we call *dynamic disconnectivity* (meaning that our dynamic networks do not necessarily have to be connected at all times). We also provide matching lower bounds, showing that all our algorithms are asymptotically *optimal* for any fixed number of leaders.

While most of the existing literature on anonymous dynamic networks relies on classical mass-distribution techniques, our work makes use of a recently introduced combinatorial structure called *history tree*, also developing its theory in new directions. Among other contributions, our results make definitive progress on two popular fundamental problems for anonymous dynamic networks: leaderless *Average Consensus* (i.e., computing the mean value of input numbers distributed among the processes) and multi-leader *Counting* (i.e., determining the exact number of processes in the network). In fact, our approach unifies and improves upon several independent lines of research on anonymous networks, including Nedić et al., IEEE Trans. Automat. Contr. 2009; Olshevsky, SIAM J. Control Optim. 2017; Kowalski–Mosteiro, ICALP 2019, SPAA 2021; Di Luna–Viglietta, FOCS 2022.

*DIAG, Sapienza University of Rome, Italy, `g.a.diluna@gmail.com`

†Department of Computer Science and Engineering, University of Aizu, Japan, `viglietta@gmail.com`

Contents

1 Introduction

Dynamic networks. An increasingly prominent area of distributed computing focuses on algorithmic aspects of *dynamic networks*, motivated by novel technologies such as wireless sensors networks, software-defined networks, networks of smart devices, and other networks with a continuously changing topology [10, 38, 41]. Typically, a network is modeled by a system of n *processes* that communicate in synchronous rounds; at each round, the network's topology changes unpredictably.

Disconnected networks. In the dynamic setting, a common assumption is that the network is *1-interval-connected*, i.e., connected at all rounds [36, 44]. However, this is not a suitable model for many real systems, due to the very nature of dynamic entities (think of P2P networks of smart devices moving unpredictably) or due to transient communication failures, which may compromise the network's connectivity. A weaker assumption is that the union of all the network's links across any T consecutive rounds induces a connected graph on the processes [34, 46]. We say that such a network is T*-union-connected*, and we call $T \geq 1$ its *dynamic disconnectivity*.[1]

Anonymous processes. Several works have focused on processes with unique IDs, which allow for efficient algorithms for many different tasks [9, 35, 36, 37, 41, 44]. However, unique IDs may not be available due to operational limitations [44] or to protect user privacy: A famous example are COVID-19 tracking apps, where assigning temporary random IDs to users was not enough to eliminate privacy concerns [50]. Systems where processes are indistinguishable are called *anonymous*. The study of *static* anonymous networks has a long history, as well [7, 8, 12, 13, 14, 27, 49, 52].

Networks with leaders. It is known that several fundamental problems for anonymous networks (a notable example being the *Counting problem*, i.e., determining the total number of processes n) cannot be solved without additional "symmetry-breaking" assumptions. The most typical choice is the presence of a single distinguished process called *leader* [2, 3, 4, 5, 23, 27, 29, 31, 40, 48, 53] or, less commonly, a subset of several leaders (and knowledge of their number[2]) [30, 32, 33, 34].

Apart from the theoretical importance of generalizing the usual single-leader scenario, studying networks with multiple leaders also has a practical impact in terms of privacy. Indeed, while the communications of a single leader can be traced, the addition of more leaders provides differential privacy for each of them.

Leaderless networks. In some networks, the presence of reliable leaders may not always be guaranteed: For example, in a mobile sensor network deployed by an aircraft, the leaders may be destroyed as a result of a bad landing; also, the leaders may malfunction during the system's lifetime. This justifies the extensive existing literature on networks with no leaders [16, 17, 42, 43, 45, 51, 54]. Notably, a large portion of works on leaderless networks have focused on the *Average Consensus problem*, where the goal is to compute the mean of a list of numbers distributed among the processes [6, 15, 16, 26, 46, 47].

1.1 Our Contributions

Summary. Focusing on anonymous dynamic networks, in this paper we completely elucidate the relationship between leaderless networks and networks with (multiple) leaders, as well as the impact

[1] We use the term "disconnected" to refer to T-union-connected networks in the sense that they may not be connected at any round. It is worth noting that non-trivial (terminating) computation requires some conditions on temporal connectivity to be met, such as a finite dynamic disconnectivity and its knowledge by all processes (refer to Proposition 2.2).

[2] It is easy to see that a network with an unknown number of leaders is equivalent to a network with no leaders at all. Also, if the leaders are distinguishable from each other, then any one of them can be elected as a unique leader. Hence, the only genuinely interesting multi-leader case is the one with a known number of indistinguishable leaders.

of the dynamic disconnectivity T on the efficiency of distributed algorithms. We remark that only a minority of existing works consider networks that are not necessarily connected at all times.

Computability. We give an exact characterization of which functions can be computed in anonymous dynamic networks with and without leaders, respectively. Namely, with at least one leader, all the so-called *multi-aggregate functions* are computable;[3] with no leaders, only the *frequency-based multi-aggregate functions* are computable[4] (see Section 2 for definitions). Interestingly, computability is independent of the dynamic disconnectivity T. Our contribution considerably generalizes a recent result on the functions computable with exactly one leader and with $T = 1$ [24].

Complete problems. While computing the so-called *Generalized Counting function* F_{GC} was already known to be a complete problem for the class of multi-aggregate functions [24], in this work we expand the picture by identifying a complete problem for the class of frequency-based multi-aggregate functions, as well: the *Frequency function* F_R (both F_{GC} and F_R are defined in Section 2). By "complete problem" we mean that computing such a function allows the immediate computation of any other function in the class with no overhead in terms of communication rounds.

Algorithms. We give efficient deterministic algorithms[5] for computing the Frequency function (Section 3) and the Generalized Counting function (Section 4). Since the two problems are complete, we automatically obtain efficient algorithms for computing *all* functions in the respective classes.

For each problem, we give two algorithms: a *terminating* version, where each process is required to commit on its output and never change it, and a *stabilizing* version, where processes are allowed to modify their outputs, provided that they eventually stabilize on the correct output.

The stabilizing algorithms for both problems run in $2Tn$ rounds regardless of the number of leaders, and do not require any knowledge of the dynamic disconnectivity T or the number of processes n. Our terminating algorithm for leaderless networks runs in $T(n + N)$ rounds with knowledge of T and an upper bound $N \geq n$; the terminating algorithm for $\ell \geq 1$ leaders runs in $(\ell^2 + \ell + 1)Tn$ rounds[6] with no knowledge about n. The latter running time is reasonable (i.e., linear) in most applications, as ℓ is typically a constant or very small compared to n.

Negative results. Some of our algorithms assume processes to have a-priori knowledge of some parameters of the network; in Section 5 we show that all of these assumptions are necessary. We also provide lower bounds that asymptotically match our algorithms' running times, assuming that the number of leaders ℓ is constant (which is a realistic assumption in most applications).

Multigraphs. All of our results hold more generally if networks are modeled as multigraphs, as opposed to the simple graphs traditionally encountered in nearly all of the literature. This is relevant in many applications: in radio communication, for instance, multiple links between processes naturally appear due to the multi-path propagation of radio waves.

1.2 Technical Advances

Our approach departs radically from the mass-distribution techniques traditionally adopted by most previous works on anonymous dynamic networks [18, 19, 20, 21, 32, 33, 34]; instead, we build upon *history trees*, a combinatorial structure recently introduced in [24]. Each node in a history tree represents a set of processes that are "indistinguishable" at a certain point in time; the number of

[3] Another way of stating this result is that it is sufficient to know the size of *any* subset of distinguished processes in order to compute all multi-aggregate functions.

[4] A similar result, limited to *static* leaderless networks, was obtained in [28].

[5] An advantage of using randomization would be the possibility of choosing unique IDs with high probability, but this would not achieve an asymptotic improvement on the running times of our algorithms; see [36].

[6] Note that the case where all processes are leaders is not equivalent to the case with no leaders, because processes do not have the information that $\ell = n$, and have to "discover" that there are no non-leader processes in the network.

such processes is the *anonymity* of the node (see Section 2 for a proper definition of history tree). The theory presented in [24] leads to an optimal solution to the Generalized Counting problem for always connected networks with a unique leader. In this paper we extend the theory to *leaderless* and *multi-leader disconnected* dynamic networks thanks to the following technical breakthroughs.

- **Main contribution.** We succeed in generalizing the theory in [24] to networks with more than one leader (Section 4.2). This is an especially challenging task because there is no obvious way of adapting the counting algorithm from [24], which critically assumes the existence of a single node in the history tree representing the leader, as well as the exact knowledge of its anonymity. In contrast, in the history tree of a multi-leader network there may be multiple nodes representing leaders, and the only information available is that the sum of their anonymities is ℓ.

  To work around this difficulty, we develop an *approximate counting procedure* which repeatedly makes "hypotheses" on the anonymity of a selected leader node. Starting from a given hypothesis, we show how to derive *conditional anonymities* for other nodes in the history tree, until we are able to estimate the network's size. Furthermore, we show that the correctness of an estimate can be verified efficiently; if the estimate is incorrect, we proceed with a new hypothesis on the anonymity of a leader's node, and so on. If carefully implemented, this strategy leads to a correct counting algorithm for multi-leader networks (Theorem 4.3).

- **Secondary contribution.** We introduce a novel technique to transform a history tree into a system of independent linear equations on anonymities (Lemma 3.1). This technique is the basis for a variety of optimal algorithms: It yields a stabilizing algorithm for the Frequency function in leaderless networks (Section 3.1), which can be converted into a terminating algorithm if a bound on the propagation time of information is known (Section 3.2), as well as a stabilizing counting algorithm for multi-leader networks (Section 4.1).

- **Additional contribution.** All of our algorithms and techniques apply to multigraphs, i.e., networks that may have multiple parallel links between pairs of processes.[7] This turns out to be a remarkably powerful feature in light of Proposition 2.3, which establishes a relationship between multigraphs and T-union-connected networks. This finding single-handedly allows us to generalize our algorithms to disconnected networks at the cost of a mere factor of T in their running times, which is worst-case optimal. This significantly improves upon previous counting algorithms for disconnected networks, which had an exponential dependence on T (see, e.g., [33]).

1.3 Impact on Fundamental Problems and State of the Art

As a byproduct of the results mentioned in Section 1.1, we are able to optimally solve two popular fundamental problems: *Generalized Counting* for multi-leader networks (because it is a multi-aggregate function) and *Average Consensus* for leaderless networks (because the mean is a frequency-based multi-aggregate function). As summarized in Table 1 and as discussed below, our results improve upon the state of the art on both problems in terms of (i) running time, (ii) assumptions on the network and the processes' knowledge, and (iii) quality of the solution. Altogether, we settle

[7]The original design of history trees is general enough to be applied to networks modeled as multigraphs [24]. However, this fact alone does not automatically imply that any algorithm that utilizes history trees should work for any network that can be modeled as a multigraph. There are tasks that can only be carried out on networks with no parallel links which can still benefit from a formulation in terms of history trees. In fact, a history tree is just an alternative way of representing a network, and making good use of this representation is entirely up to the algorithm.

open problems from ICALP 2019 [30], SPAA 2021 [32], and FOCS 2022 [24]. For a more thorough discussion and a comprehensive survey of related literature, refer to Appendix B.

Average Consensus. This problem has been studied for decades by the distributed control and distributed computing communities [6, 15, 16, 17, 32, 42, 45, 47, 51, 54]. In the following, we argue that our results directly improve upon the current state of the art on this problem. A more detailed discussion can be found in the surveys [26, 43, 46] and in Appendix B.

A convergent algorithm with a running time of $O\left(Tn^3 \log(1/\epsilon)\right)$ is given in [42]. The algorithm works in T-union-connected networks with no knowledge of T, but it rests on the assumption that the degree of each process in the network has a known upper bound. Assuming an always connected network, [16] gives an algorithm that converges in $O\left(n^4 \log(n/\epsilon)\right)$ rounds. We remark that both algorithms are only ϵ-convergent; therefore, not only does our stabilizing algorithm improve upon their running times, but it solves a more difficult problem under weaker assumptions.[8]

The algorithm in [15] stabilizes to the actual average in a linear number of rounds, but it is a randomized Monte Carlo algorithm and requires the network to be connected at each round. In contrast, our linear-time stabilizing algorithm is deterministic and works in disconnected networks.

As for terminating algorithms, the one in [32] terminates in $O\left(n^5 \log^3(n)/\ell\right)$ rounds assuming the presence of a known number ℓ of leaders and an always connected network. Since the number of leaders is known, our terminating algorithm for Generalized Counting also solves Average Consensus with a running time that improves upon [32] and does not require the network to be connected. We remark that our algorithm terminates in linear time when ℓ is constant.

Generalized Counting. Our results on this problem are direct generalizations of [24] to the case of multiple leaders and disconnected networks. The best previous counting algorithm with multiple known leaders is the one in [34], which terminates in $O\left(n^4 \log^3(n)/\ell\right)$ rounds and assumes the network to be connected at each round. In the same setting, our stabilizing and terminating algorithms have running times of $2n$ rounds and $(\ell^2 + \ell + 1)n$ rounds, respectively.

The only other result for disconnected networks is the recent preprint [33], which gives an algorithm that terminates in $\widetilde{O}\left(n^{2T+3}/\ell\right)$ rounds using $O(\log n)$-sized messages. Our terminating algorithm has a linear dependence on both n and T, which is an exponential improvement upon the running time of [33], but it requires polynomial-sized messages.

2 Definitions and Preliminaries

We will give preliminary definitions and results, and recall some properties of history trees from [24].

Processes and networks. A *dynamic network* is modeled by an infinite sequence $\mathcal{G} = (G_t)_{t\geq 1}$, where $G_t = (V, E_t)$ is an undirected multigraph whose vertex set $V = \{p_1, p_2, \ldots, p_n\}$ is a system of n *anonymous processes* and E_t is a multiset of edges representing *links* between processes.

Each process p_i starts with an *input* $\lambda(p_i)$, which is assigned to it at *round* 0. It also has an internal state, which is initially determined by $\lambda(p_i)$. At each *round* $t \geq 1$, every process composes a message (depending on its internal state) and broadcasts it to its neighbors in G_t through all its incident links.[9] By the end of round t, each process reads all messages coming from its neighbors

[8] More generally, averaging algorithms based on Metropolis rules cannot be applied to our model, because they require all processes to know their out-degree before the broadcast phase of each round. This was noted by Charron-Bost et al. in [16]: "Unfortunately, local algorithms cannot implement the Metropolis rule over dynamic networks. [...] an agent's next estimate $x_i(t)$ depends on information present *within distance* 2 of agent i in the communication graph $\mathbb{G}(t)$, which is not local *enough* when the network is subject to change".

[9] In order to model wireless radio communication, it is natural to assume that each process in a dynamic network broadcasts its messages to all its neighbors (a message is received by anyone within communication range). The network's anonymity prevents processes from specifying single destinations.

Problem	*Reference*	*Leaders*	*Disconn.*	*Term.*	*Notes*	*Running time*
Average Consensus	[42]	$\ell = 0$	✓		ϵ-convergence, T unknown, upper bound on processes' degrees known	$O(Tn^3 \log(1/\epsilon))$
	[16]	$\ell = 0$			ϵ-convergence	$O(n^4 \log(n/\epsilon))$
	[15]	$\ell = 0$			randomized Monte Carlo	$O(n)$
	[32]	$\ell \geq 1$		✓	ℓ known	$O(n^5 \log^3(n)/\ell)$
	this work	$\ell = 0$	✓		T unknown	$2Tn$
		$\ell = 0$	✓	✓	T and $N \geq n$ known	$T(n+N)$
(Generalized) Counting	[24]	$\ell = 1$				$2n-2$
		$\ell = 1$		✓		$3n-2$
	[34]	$\ell \geq 1$		✓	ℓ known	$O(n^4 \log^3(n)/\ell)$
	[33]	$\ell \geq 1$	✓	✓	ℓ and T known, $O(\log n)$-size messages	$\widetilde{O}(n^{2T+3}/\ell)$
	this work	$\ell \geq 1$	✓		ℓ known, T unknown	$2Tn$
		$\ell \geq 1$	✓	✓	ℓ and T known	$(\ell^2 + \ell + 1)Tn$

Table 1: Comparing results for Average Consensus and Counting in anonymous dynamic networks. For algorithms that support disconnected networks, T indicates the dynamic disconnectivity.

and updates its internal state according to a local algorithm $\mathcal{A}$. Note that $\mathcal{A}$ is deterministic and is the same for all processes.[10] The input of each process also includes a *leader flag*. The processes whose leader flag is set are called *leaders* (or *supervisors*). We will denote the number of leaders as ℓ.

Each process also returns an *output* at the end of each round, which is determined by its current internal state. A system is said to *stabilize* if the outputs of all its processes remain constant from a certain round onward; note that a process' internal state may still change even when its output is constant. A process may also decide to explicitly *terminate* and no longer update its internal state. When all processes have terminated, the system is said to *terminate*, as well.

We say that $\mathcal{A}$ *computes* a function F if, whenever the processes are assigned inputs $\lambda(p_1)$, $\lambda(p_2)$, $\dots$, $\lambda(p_n)$ and all processes execute the local algorithm $\mathcal{A}$ at every round, the system eventually stabilizes with each process p_i giving the desired output $F(p_i, \lambda)$.[11] A stronger notion of computation requires the system to not only stabilize but also to explicitly terminate with the correct output. The (worst-case) *running time* of $\mathcal{A}$, as a function of n, is the maximum number of rounds it takes for the system to stabilize (and optionally terminate), taken across all possible dynamic networks of size n and all possible input assignments.

Classes of functions. Let $\mu_\lambda = \{(z_1, m_1), (z_2, m_2), \dots, (z_k, m_k)\}$ be the multiset of all processes' inputs. That is, for all $1 \leq i \leq k$, there are exactly m_i processes $p_{j_1}, p_{j_2}, \dots, p_{j_{m_i}}$ whose input is $z_i = \lambda(p_{j_1}) = \lambda(p_{j_2}) = \dots = \lambda(p_{j_{m_i}})$; note that $n = \sum_{i=1}^{k} m_i$. A *multi-aggregate* function is defined as a function F of the form $F(p_i, \lambda) = \psi(\lambda(p_i), \mu_\lambda)$, i.e., such that the output of each process depends only on its own input and the multiset of all processes' inputs.

The special multi-aggregate functions $F_C(p_i, \lambda) = n$ and $F_{GC}(p_i, \lambda) = \mu_\lambda$ are called the *Counting* function and the *Generalized Counting* function, respectively. It is known that, if a system can compute the Generalized Counting function F_{GC}, then it can compute any multi-aggregate function

[10]This network model bears some similarities with Population Protocols, although there are radical differences in the way symmetry is handled. The point-to-point communication model of Population Protocols automatically breaks the symmetry between communicating agents, greatly simplifying problems such as Leader Election, Average Consensus, etc.

[11]Formally, a function computed by a system of n processes maps n-tuples of input values to n-tuples of output values. Writing such a function as $F(p_i, \lambda)$ emphasizes that the output of a process may depend on all processes' inputs, as well as on the process itself. That is, different processes may give different outputs.

in the same number of rounds: thus, F_{GC} is *complete* for the class of multi-aggregate functions [24].

For any $\alpha \in \mathbb{R}^+$, we define $\alpha \cdot \mu_\lambda$ as $\{(z_1, \alpha \cdot m_1), (z_2, \alpha \cdot m_2), \dots, (z_k, \alpha \cdot m_k)\}$. We say that a multi-aggregate function $F(p_i, \lambda) = \psi(\lambda(p_i), \mu_\lambda)$ is *frequency-based* if $\psi(z, \mu_\lambda) = \psi(z, \alpha \cdot \mu_\lambda)$ for every positive integer α and every input z (see [28]). That is, F depends only on the "frequency" of each input in the system, rather than on their actual multiplicities. Notable examples include statistical functions such as mean, variance, maximum, median, mode, etc. The problem of computing the mean of all input values is called *Average Consensus* [6, 15, 16, 17, 26, 32, 42, 43, 45, 46, 47, 51, 54].

The frequency-based multi-aggregate function $F_R(p_i, \lambda) = \frac{1}{n} \cdot \mu_\lambda$ is called *Frequency* function, and is complete for the class of frequency-based multi-aggregate functions, as shown below.

Proposition 2.1. *If F_R can be computed (with termination), then all frequency-based multi-aggregate functions can be computed (with termination) in the same number of rounds, as well.*

Proof. Suppose that a process p_i has determined $\frac{1}{n} \cdot \mu_\lambda = \{(z_1, m_1/n), (z_2, m_2/n), \dots, (z_k, m_k/n)\}$. Then it can immediately find the smallest integer $d > 0$ such that $d \cdot (m_i/n)$ is an integer for all $1 \leq i \leq k$. Note that $\frac{d}{n} \cdot \mu_\lambda$ is a multiset. Hence, in the same round, p_i can compute any desired function $\psi(\lambda(p_i), \frac{d}{n} \cdot \mu_\lambda)$, and thus any frequency-based multi-aggregate function, by definition. $\square$

History trees. *History trees* were introduced in [24] as a tool of investigation for anonymous dynamic networks; an example is found in Figure 1. A history tree is a representation of a dynamic network given some inputs to the processes. It is an infinite graph whose nodes are partitioned into *levels* L_t, with $t \geq -1$; each node in L_t represents a class of processes that are *indistinguishable* at the end of round t (with the exception of L_{-1}, which contains a single node r representing all processes). The definition of distinguishability is inductive: at the end of round 0, two processes are distinguishable if and only if they have different inputs. At the end of round $t \geq 1$, two processes are distinguishable if and only if they were already distinguishable at round $t-1$ or if they have received different multisets of messages at round t.

Each node in level L_0 has a label indicating the input of the processes it represents. There are also two types of edges connecting nodes in adjacent levels. The *black edges* induce an infinite tree rooted at node $r \in L_{-1}$ which spans all nodes. The presence of a black edge $\{v, v'\}$, with $v \in L_t$ and $v' \in L_{t+1}$, indicates that the *child node* v' represents a subset of the processes represented by the *parent node* v. The *red multi-edges* represent communications between processes. The presence of a red edge $\{v, v'\}$ with multiplicity m, with $v \in L_t$ and $v' \in L_{t+1}$, indicates that, at round $t+1$, each process represented by v' receives m (identical) messages from processes represented by v.

As time progresses and processes exchange messages, they are able to locally construct finite portions of the history tree. In [24], it is shown that there is a local algorithm $\mathcal{A}^*$ that allows each process to locally construct and update its own *view* of the history tree at every round. The view of a process p at round $t \geq 0$ is the subgraph of the history tree which is spanned by all the shortest paths (using black and red edges indifferently) from the root r to the node in L_t representing p (see Figure 1). As proved in [24, Theorem 3.1], the view of a process at round t contains all the information that the process may be able to use at that round. This justifies the convention that all processes always execute $\mathcal{A}^*$, constructing their local view of the history tree and broadcasting (a representation of) it at every round, regardless of their task. Then, they simply compute their task-dependent outputs as a function of their respective views.

We define the *anonymity* of a node v of the history tree as the number of processes that v represents, and we denote it as $a(v)$. It follows that $\sum_{v \in L_t} a(v) = n$ for all $t \geq -1$, and that the anonymity of a node is equal to the sum of the anonymities of its children. Naturally, a process is not aware of the anonymities of the nodes in its view of the history tree, unless it can somehow infer them from the view's structure itself. In fact, computing the Generalized Counting function is

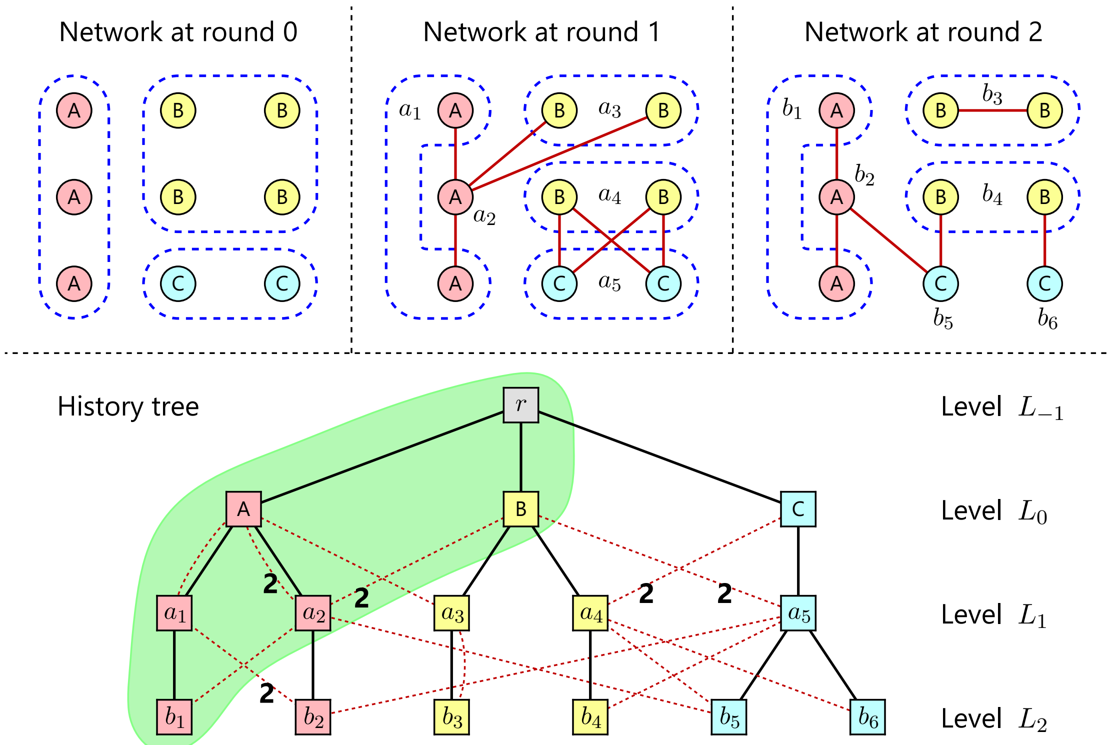

Figure 1: The first rounds of a dynamic network with $n = 9$ processes and the corresponding levels of the history tree. Level L_t consists of all nodes at distance $t+1$ from the root r. The multiplicities of the red multi-edges of the history tree are explicitly indicated only when greater than 1. The letters A, B, C denote processes' inputs; all other labels have been added for the reader's convenience, and indicate classes of indistinguishable processes (non-trivial classes are also indicated by dashed blue lines). Note that the two processes in b_4 are still indistinguishable at the end of round 2, although they are linked to the distinguishable processes b_5 and b_6. This is because such processes were in the same class a_5 at round 1. The subgraph in the green blob is the *view* of the two processes in b_1.

equivalent to determining the anonymities of all the nodes in L_0. Similarly, computing the Frequency function corresponds to determining the value $a(v)/n$ for all $v \in L_0$.

Computation in disconnected networks. Although the network G_t at each individual round may be disconnected, we assume the dynamic network to be T*-union-connected*. That is, there is a *dynamic disconnectivity* parameter $T \geq 1$ such that the sum of any T consecutive G_t's is a connected multigraph.[12] Thus, for all $i \geq 1$, the multigraph $\left(V, \bigcup_{t=i}^{i+T-1} E_t\right)$ is connected (we remark that a union of multisets adds together the multiplicities of equal elements).[13]

Proposition 2.2. *Any non-trivial function is impossible to compute with termination unless the processes have some knowledge about T. (A function is "trivial" if it can be computed locally.)*

Proof. A function $F(p_i, \lambda)$ is trivial if and only if it is of the form $F(p_i, \lambda) = \psi(\lambda(p_i))$, i.e., the output of any process p_i depends only on its input $\lambda(p_i)$, and not on the inputs of other processes. Assume for a contradiction that a non-trivial function $F(p_i, \lambda)$ is computed with termination by an algorithm $\mathcal{A}$ with no knowledge of T.

Since F is non-trivial, there is an input value z and two distinct output values y and y' with the following properties. (i) If p_1 is the only process in a network (i.e., $n = 1$), and p_1 is assigned input z and executes $\mathcal{A}$, it terminates in t rounds with output y. (ii) There exists a network size $\tilde{n} > 1$ and an input assignment λ with $\lambda(p_1) = z$ such that, whenever the processes in a network of size $\tilde{n}$ are assigned input λ and execute $\mathcal{A}$, the process p_1 eventually terminates with output $y' \neq y$.

Let us now consider a dynamic network of $\tilde{n}$ processes, where p_1 is kept disconnected from all other processes for the first t rounds (hence $T > t$). Assign input λ to the processes, and let them execute algorithm $\mathcal{A}$. Due to property (i), since p_1 is isolated for t rounds and has no knowledge of T, it terminates in t rounds with output y. This contradicts property (ii), which states that p_1 should terminate with output $y' \neq y$. □

Proposition 2.3. *A function F can be computed (with termination) within $f(n)$ rounds in any dynamic network with $T = 1$ if and only if F can be computed (with termination) within $T \cdot f(n)$ rounds in any dynamic network with $T \geq 1$, assuming that T is known to all processes.*

Proof. Subdivide time into *blocks* of T consecutive rounds, and consider the following algorithm. Each process collects and stores all messages it receives within a same block, and updates its state all at once at the end of the block. This reduces any T-union-connected network $\mathcal{G} = ((V, E_t))_{t\geq 1}$ to a 1-union-connected network $\mathcal{G}' = ((V, E'_t))_{t\geq 1}$, where $E'_t = \bigcup_{i=(t-1)T+1}^{tT} E_i$. Thus, if F can be computed within $f(n)$ rounds in all 1-union-connected networks (which include $\mathcal{G}'$), then F can be computed within $Tf(n)$ rounds in the original network $\mathcal{G}$.[14]

Conversely, consider a 1-union-connected network $\mathcal{G}$, and construct a T-union-connected network $\mathcal{G}'$ by inserting $T - 1$ empty rounds (i.e., rounds with no links at all) between every two consecutive rounds of $\mathcal{G}$. Since no information circulates during the empty rounds, if F cannot be computed within $f(n)$ rounds in $\mathcal{G}$, then F cannot be computed within $Tf(n)$ rounds in $\mathcal{G}'$ (recall that running times are measured in the worst case across all possible networks). □

[12] By definition, the sum of (multi-)graphs is obtained by adding together their adjacency matrices.

[13] Our T-union-connected networks should not be confused with the T-*interval*-connected networks from [36]. In those networks, the *intersection* (as opposed to the union) of any T consecutive E_t's induces a connected (multi)graph. In particular, a T-interval-connected network is connected at every round, while a T-union-connected network may not be, unless $T = 1$. Incidentally, a network is 1-interval-connected if and only if it is 1-union-connected.

[14] Note that this argument is correct because algorithms are required to work for all multigraphs, as opposed to simple graphs only. Indeed, since a process p_i may receive multiple messages from the same process p_j within a same block, the resulting network $\mathcal{G}'$ may have multiple links between p_i and p_j in a same round, even if $\mathcal{G}$ does not.

Relationship with the dynamic diameter. A concept closely related to the dynamic disconnectivity T of a network is its *dynamic diameter* (or *temporal diameter*) D, which is defined as the maximum number of rounds it may take for information to travel from any process to any other process at any point in time [10, 38]. It is a simple observation that $T \leq D \leq T(n-1)$.

We chose to use T, as opposed to D, to measure the running times of our algorithms for several reasons. Firstly, T is well defined (i.e., finite) if and only if D is; however, T has a simpler definition, and is arguably easier to directly estimate or enforce in a real network. Secondly, Proposition 2.3, as well as all of our theorems, remain valid if we replace T with D; nonetheless, stating the running times of our algorithms in terms of T is better, because $T \leq D$.

3 Computation in Leaderless Networks

We will give a stabilizing and a terminating algorithm that efficiently compute the Frequency function F_R in all leaderless networks with finite dynamic disconnectivity T. As a consequence, *all* frequency-based multi-aggregate functions are efficiently computable as well, due to Proposition 2.1. Moreover, Proposition 5.1 states that no other functions are computable in leaderless networks, and Proposition 5.2 shows that our algorithms are asymptotically optimal.

3.1 Stabilizing Algorithm

We will use the procedure in Listing 1 as a subroutine in some of our algorithms. Its purpose is to construct a homogeneous system of $k-1$ independent linear equations[15] involving the anonymities of all the k nodes in a level of a process' view. We will first give some definitions.

In (a view of) a history tree, if a node $v \in L_t$ has exactly one child (i.e., there is exactly one node $v' \in L_{t+1}$ such that $\{v, v'\}$ is a black edge), we say that v is *non-branching*. We say that two non-branching nodes $v_1, v_2 \in L_t$, whose respective children are $v'_1, v'_2 \in L_{t+1}$, are *exposed* with multiplicity (m_1, m_2) if the red edges $\{v'_1, v_2\}$ and $\{v'_2, v_1\}$ are present with multiplicities $m_1 \geq 1$ and $m_2 \geq 1$, respectively. A *strand* is a path $(w_1, w_2, \dots, w_k)$ in (a view of) a history tree consisting of non-branching nodes such that, for all $1 \leq i < k$, the node w_i is the parent of w_{i+1}. We say that two strands P_1 and P_2 are *exposed* if there are two exposed nodes $v_1 \in P_1$ and $v_2 \in P_2$.

Intuitively, the procedure in Listing 1 searches for a long-enough sequence of levels in the given view $\mathcal{V}$, say from L_s to L_t, where all nodes are non-branching. That is, the nodes in $L_s \cup L_{s+1} \cup \dots \cup L_t$ can be partitioned into $k = |L_s| = |L_t|$ strands. Then the procedure searches for pairs of exposed strands, each of which yields a linear equation involving the anonymities of some nodes of L_t, until it obtains $k-1$ linearly independent equations.[16] Note that the search may fail (in which case Listing 1 returns $t = -1$) or it may produce incorrect equations. The following lemma specifies sufficient conditions for Listing 1 to return a correct and non-trivial system of equations for some $t \geq 0$.

Lemma 3.1. *Let $\mathcal{V}$ be the view of a process in a T-union-connected network of size n taken at round t', and let Listing 1 return (t, S) on input $\mathcal{V}$. Assume that one of the following conditions holds:*

1. *$t \geq 0$ and $t' \geq t + Tn$, or*
2. *$t' \geq 2Tn$.*

[15] A linear system is *homogeneous* if all its constant terms are zero.

[16] The reason why we have to consider strands spanning several levels of the history tree (as opposed to looking at a single level) is that the dynamic disconnectivity T is not known, and thus Proposition 2.3 cannot be applied directly.

Listing 1: Constructing a system of equations in the anonymities of some nodes in a view.

```
1  # Input: a view $\mathcal{V}$ with levels $L_{-1}$, $L_0$, $L_1$, ..., $L_h$
2  # Output: $(t, S)$, where $t$ is an integer and $S$ is a system of linear equations
3
4  Assign $s := 0$
5  For $t := 0$ to $h$
6     If $L_t$ contains a node with no children, return $(-1, \emptyset)$
7     If $L_t$ contains a node with more than one child, assign $s := t + 1$
8     Else
9        Let $k = |L_s| = |L_t|$ and let $u_i$ be the $i$th node in $L_t$
10       Let $P_i$ be the strand starting in $L_s$ and ending in $u_i \in L_t$
11       Let $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$
12       Let $G$ be the graph on $\mathcal{P}$ whose edges are pairs of exposed strands
13       If $G$ is connected
14          Let $G' \subseteq G$ be any spanning tree of $G$
15          Assign $S := \emptyset$
16          For each edge $\{P_i, P_j\}$ of $G'$
17             Find any two exposed nodes $v_1 \in P_i$ and $v_2 \in P_j$
18             Let $(m_1, m_2)$ be the multiplicity of the exposed pair $(v_1, v_2)$
19             Add to $S$ the equation $m_1 x_i = m_2 x_j$
20          Return $(t, S)$
```

Then, $0 \le t \le Tn$, and S is a homogeneous system of $k-1$ independent linear equations (with integer coefficients) in $k = |L_t|$ variables x_1, x_2, ..., x_k. Moreover, S is satisfied by assigning to x_i the anonymity of the ith node of L_t, for all $1 \le i \le k$.

Proof. It is well known that, if $T = 1$, information takes less than n rounds to travel from a process to any other process [36]. Thus, if $T \ge 1$, it takes less than Tn rounds (cf. Proposition 2.3). Since $\mathcal{V}$ is a view taken at round $2Tn$ (or after), all levels of $\mathcal{V}$ up to $L_{t'-Tn+1}$ are *complete*, i.e., all nodes in the first $t' - Tn + 1$ levels of the network's history tree also appear in the view $\mathcal{V}$.

Assume Condition 2 first. Since $t' \ge 2Tn$, all levels of $\mathcal{V}$ up to L_{Tn+1} are complete. Since the anonymity of the root of $\mathcal{V}$ is n, there must be less than n branching nodes in $\mathcal{V}$. Therefore, the first Tn levels contain an interval of at least T consecutive levels, say from L_r to L_{r+T-1}, where all nodes are non-branching and can be partitioned into $|L_r| = |L_{r+T-1}|$ strands P_i.

Note that a link between two processes at any round r' in the interval $[r+1, r+T]$ determines a pair of exposed nodes in $L_{r'-1}$. Thus, by definition of T-union-connected network, the graph of exposed strands between L_r and L_{r+T-1} (constructed as G in Line 12) is connected. It follows that the execution of Listing 1 terminates at Line 20 (as opposed to Line 6), at the latest when $t = r + T - 1$. Thus, the procedure returns a pair (t, S) with $0 \le t \le r + T - 1 \le Tn$. In particular, all levels of $\mathcal{V}$ up to L_{t+1} are complete.

Now assume Condition 1. Since $t' \ge t + Tn$, all levels of $\mathcal{V}$ up to L_{t+1} are complete in this case, as well. Since $t \ge 0$ by assumption, the execution of Listing 1 terminates at Line 20. The termination condition is met when long-enough strands are found; as proved above, this must happen while $t \le Tn$.

We have proved that, in both cases, the inequalities $0 \le t \le Tn$ hold, and all levels of $\mathcal{V}$ up to L_{t+1} are complete. Let us now examine the linear system S. Observe that S is homogeneous because it consists of homogeneous linear equations (cf. Line 19). Also, since the spanning tree G' constructed in Line 14 has $k-1$ edges, S contains $k-1$ equations. We will prove that they are linearly independent by induction on k. If $k = 1$, there is nothing to prove. Otherwise, let P_i

be a leaf of G', and let $\{P_i, P_j\}$ be its incident edge. Then, S contains an equation Q of the form $m_1x_i = m_2x_j$ with $m_1m_2 \neq 0$. Let S' be the system obtained by removing Q from S; equivalently, S' corresponds to the tree obtained by removing the leaf P_i from G'. By the inductive hypothesis, no linear combination of equations in S' yields $0 = 0$. On the other hand, if Q is involved in a linear combination with a non-zero coefficient, then the variable x_i cannot vanish, because it only appears in Q. Therefore, the equations in S are independent.

It remains to prove that a solution to S is given by the anonymities of the nodes of L_t. It was shown in [24, Lemma 4.1] that, if v_1 and v_2 are exposed in $\mathcal{V}$, as well as in the history tree containing $\mathcal{V}$, with multiplicity (m_1, m_2), then $m_1a(v_1) = m_2a(v_2)$.[17] To conclude our proof, it is sufficient to note that, since the nodes of a strand P_i are non-branching in $\mathcal{V}$ as well as in the underlying history tree (recall that all levels of $\mathcal{V}$ up to L_{t+1} are complete), they all have the same anonymity, which is the anonymity of the ending node $w_i \in L_t$. □

Theorem 3.2. *There is an algorithm that computes F_R in all T-union-connected anonymous networks with no leader and stabilizes in at most $2Tn$ rounds, assuming no knowledge of T or n.*

Proof. Our local algorithm is as follows. Run Listing 1 on the process' view $\mathcal{V}$, obtaining a pair (t, S). If $t = -1$ or S is not a homogeneous system of $k-1$ independent linear equations in k variables, output "Unknown". Otherwise, since the rank of the coefficient matrix of S is $k-1$, the general solution to S has exactly one free parameter, due to the Rouché–Capelli theorem. Therefore, by Gaussian elimination, it is possible to express every variable x_i as a rational multiple of x_1, i.e., $x_i = \alpha_i x_1$ for some $\alpha_i \in \mathbb{Q}^+$ (note that the coefficients of S are integers). Let $L_t = \{w_1, w_2, \dots, w_k\}$ and $L_0 = \{v_1, v_2, \dots, v_{k'}\}$. For every node $v_i \in L_0$, define $\beta_i \in \mathbb{Q}^+$ as $\beta_i = \sum_{w_j \in L_t \text{ descendant of } v_i} \alpha_j$, and let $\beta = \sum_i \beta_i$. Then, output

$$\{(\text{label}(v_1), \beta_1/\beta), (\text{label}(v_2), \beta_2/\beta), \dots, (\text{label}(v_{k'}), \beta_{k'}/\beta)\}.$$

The correctness and stabilization time of the above algorithm directly follow from Lemma 3.1. Specifically, at any round $t' \geq 2Tn$, Condition 2 of Lemma 3.1 is met, and the system S is satisfied by the anonymities of the nodes in L_t. Thus, $a(v_i) = \alpha_i a(v_1)$ for all $v_i \in L_0$, and therefore $\beta_i/\beta = a(v_i)/n$. We conclude that, for any input assignment λ, the algorithm stabilizes on the correct output $\frac{1}{n} \cdot \mu_\lambda$ within $2Tn$ rounds. □

3.2 Terminating Algorithm

We will now give a certificate of correctness that can be used to turn the stabilizing algorithm of Theorem 3.2 into a terminating algorithm. The certificate relies on a-priori knowledge of the dynamic disconnectivity T and an upper bound N on the size of the network n; these assumptions are justified by Proposition 2.2 and Proposition 5.3, respectively.

Theorem 3.3. *There is an algorithm that computes F_R in all T-union-connected anonymous networks with no leader and terminates in at most $T(n + N)$ rounds, assuming that T and an upper bound $N \geq n$ are known to all processes.*[18]

Proof. Our terminating algorithm is as follows. Run Listing 1 on the process' view $\mathcal{V}$, obtaining a pair (t, S), and then do the same computations as in the algorithm in Theorem 3.2. If $t \geq 0$ and the current round t' satisfies $t' \geq t + TN$, then the output is correct, and the process terminates.

[17] If v_1 and v_2 are exposed in $\mathcal{V}$ but not in the underlying history tree, then they have some children not in $\mathcal{V}$, and therefore the equation may not hold.

[18] If the dynamic diameter D of the network is known, the termination time improves to $Tn + D$ rounds.

The correctness of this algorithm is a direct consequence of Lemma 3.1. Indeed, the algorithm only terminates when $t \geq 0$ and $t' \geq t + TN \geq t + Tn$, and hence it gives the correct output because Condition 1 of Lemma 3.1 is met. As for the running time, assume that $t' = T(n+N) \geq 2Tn$. Since Condition 2 of Lemma 3.1 is met, we have $0 \leq t \leq Tn$. Thus, $t' = T(n+N) \geq t + Tn$, and the algorithm terminates at round t'. □

4 Computation in Networks with Leaders

We will give a stabilizing and a terminating algorithm that efficiently compute the Generalized Counting function F_{GC} in all networks with $\ell \geq 1$ leaders and finite dynamic disconnectivity T. Therefore, *all* multi-aggregate functions are efficiently computable as well, due to [24, Theorem 2.1]. Moreover, Proposition 5.4 states that no other functions are computable in networks with leaders, and Proposition 5.6 shows that our algorithms are asymptotically optimal for any fixed $\ell \geq 1$.

4.1 Stabilizing Algorithm

We will once again make use of the subroutine in Listing 1, this time assuming that the number of leaders $\ell \geq 1$ is known to all processes. This assumption is justified by Proposition 5.5.

Theorem 4.1. *There is an algorithm that computes F_{GC} in all T-union-connected anonymous networks with $\ell \geq 1$ leaders and stabilizes in at most $2Tn$ rounds, assuming that ℓ is known to all processes, but assuming no knowledge of T or n.*

Proof. The algorithm proceeds as in Theorem 3.2. When the fractions $\beta_1, \beta_2, \dots, \beta_{k'}$ have been computed, as well as their sum β, we perform the following additional steps. Let $L_0 = \{v_1, v_2, \dots, v_{k'}\}$, and let $\{v_{j_1}, v_{j_2}, \dots, v_{j_l}\} \subseteq L_0$ be the set of nodes in L_0 representing leader processes,[19] i.e., such that $\text{label}(v_{j_i})$ has the leader flag set for all $1 \leq i \leq l$. Compute $\beta' = \sum_{i=1}^{l} \beta_{j_i}$ and $\gamma_i = \ell\beta_i/\beta'$ for all $1 \leq i \leq k'$, and output

$$\{(\text{label}(v_1), \gamma_1), (\text{label}(v_2), \gamma_2), \dots, (\text{label}(v_{k'}), \gamma_{k'})\}.$$

The correctness follows from the fact that, as shown in Theorem 3.2, at any round $\geq 2Tn$ we have $\beta_i/\beta = a(v_i)/n$ for all $1 \leq i \leq k'$. Adding up these equations for all $i \in \{j_1, j_2, \dots, j_l\}$, we obtain $\beta'/\beta = \ell/n$, and therefore $n = \ell\beta/\beta'$. We conclude that

$$\gamma_i = \frac{\ell\beta_i}{\beta'} = \frac{\ell\beta\beta_i}{\beta'\beta} = \frac{n\beta_i}{\beta} = a(v_i).$$

Thus, within $2Tn$ rounds, the algorithm stably outputs the anonymities of all nodes in L_0. As observed in Section 2, this is equivalent to computing the Generalized Counting function F_{GC}. □

4.2 Terminating Algorithm

We will now present the main result of this paper. As already remarked, giving an efficient certificate of correctness for the (Generalized) Counting problem with multiple leaders is a highly non-trivial task for which a radically different approach is required. Note that it is not possible to simply adapt the single-leader algorithm in [24] by setting the anonymity of the leader node in the history tree to ℓ instead of 1. Indeed, as soon as some leaders get disambiguated, the leader node splits into

[19] In general we have $l \leq \ell$, because some nodes of L_0 may represent more than one leader.

several children nodes whose anonymities are unknown (we only know that their sum is ℓ). There is no way around this difficulty other than developing a new technique.

The subroutine `ApproxCount`. We first introduce the subroutine `ApproxCount`, whose formal description and proof of correctness are found in Appendix A. The purpose of `ApproxCount` is to compute an approximation n' of the total number of processes n (or report various types of failure). It takes as input a view $\mathcal{V}$ of a process, the number of leaders ℓ, and two integer parameters s and x, representing the index of a level of $\mathcal{V}$ and the anonymity of a leader node in L_s, respectively.

Discrepancy δ. Suppose that `ApproxCount` is invoked with arguments $\mathcal{V}$, s, x, ℓ, where $1 \le x \le \ell$, and let τ be the first leader node in level L_s of $\mathcal{V}$ (if τ does not exist, the procedure immediately returns the error code $n' = -1$). We define the *discrepancy* δ as the ratio $x/a(\tau)$. Clearly, $\delta \le \ell$. Note that, since $a(\tau)$ is not a-priori known by the process executing `ApproxCount`, then neither is δ.

Conditional anonymities. `ApproxCount` starts by assuming that the anonymity of τ is x, and makes deductions on other anonymities based on this assumption. Thus, we will distinguish between the actual anonymity of a node $a(v)$ and the *conditional anonymity* $a'(v) = \delta a(v)$ that `ApproxCount` may compute under the initial assumption that $a'(\tau) = x = \delta a(\tau)$.

Overview of `ApproxCount`. The procedure `ApproxCount` scans the levels of $\mathcal{V}$ starting from L_s, making "guesses" on the conditional anonymities of nodes based on already known conditional anonymities. Generalizing some lemmas from [24], we develop a criterion to determine when a guess is correct. This yields more nodes with known conditional anonymities, and therefore more guesses (the details are in Appendix A). As soon as it has obtained enough information, the procedure stops and returns (n', t), where L_t is the level scanned thus far. If the information gathered satisfies certain criteria, then n' is an approximation of n. Otherwise, n' is an error code, as detailed below.

Error codes. If L_s contains no leader nodes, the procedure returns the error code $n' = -1$. If, before gathering enough information on n, the procedure encounters a descendant of τ with more than one child in $\mathcal{V}$, it returns the error code $n' = -2$. If it determines that the conditional anonymity of a node is not an integer, it returns the error code $n' = -3$. Finally, if it determines that the sum ℓ' of the conditional anonymities of the leader nodes is not ℓ, it returns $n' = -1$ if $\ell' < \ell$ and $n' = -3$ if $\ell' > \ell$.

Correctness of `ApproxCount`. The following lemma gives some conditions that guarantee that `ApproxCount` has the expected behavior; it also gives some bounds on the number of rounds it takes for `ApproxCount` to produce an approximation n' of n, as well as a criterion to determine if $n' = n$. The lemma's proof is rather lengthy and technical, and is deferred to Appendix A.

Lemma 4.2. *Let* `ApproxCount`$(\mathcal{V}, s, x, \ell)$ *return* (n', t). *Assume that* τ *exists and* $x \ge a(\tau)$. *Let* τ' *be the (unique) descendant of* τ *in* $\mathcal{V}$ *at level* L_t, *and let* $L_{t'}$ *be the last level of* $\mathcal{V}$. *Then:*

(i) If $x = a(\tau) = a(\tau')$, *then* $n' \neq -3$.

(ii) If $n' > 0$ *and* $t' \ge t + n'$ *and* $a(\tau) = a(\tau')$, *then* $n' = n$.

(iii) If $t' \ge s + (\ell+2)n - 1$, *then* $s \le t \le s + (\ell+1)n - 1$ *and* $n' \neq -1$. *Moreover, if* $n' = -2$, *then* L_t *contains a leader node with at least two children in* $\mathcal{V}$. □

Our terminating algorithm assumes that all processes know the number of leaders $\ell \ge 1$ and the dynamic disconnectivity T. Again, this is justified by Proposition 5.5 and Proposition 2.2.

Theorem 4.3. *There is an algorithm that computes* F_{GC} *in all* T*-union-connected anonymous networks with* $\ell \ge 1$ *leaders and terminates in at most* $(\ell^2 + \ell + 1)Tn$ *rounds, assuming that* ℓ *and* T *are known to all processes, but assuming no knowledge of* n.

Listing 2: Solving the Counting problem with $\ell \geq 1$ leaders.

```
1  # Input: a view 𝒱 and a positive integer ℓ
2  # Output: either a positive integer n or "Unknown"
3
4  Assign n* := −1 and s := 0 and c := 0
5  Let b be the number of leader branches in 𝒱
6  While c ≤ ℓ − b
7     Assign t* := −1
8     For x := ℓ downto 1
9        Assign (n′, t) := ApproxCount(𝒱, s, x, ℓ)        # see Listing 3 in Appendix A
10       Assign t* := max{t*, t}
11       If n′ = −1, return "Unknown"
12       If n′ = −2, break out of the for loop
13       If n′ > 0
14          If n* = −1, assign n* := n′
15          Else if n* ≠ n′, return "Unknown"
16          Assign c := c + 1 and break out of the for loop
17    Assign s := t* + 1
18 Let L_{t′} be the last level of 𝒱
19 If t′ ≥ t* + n*, return n*
20 Else return "Unknown"
```

Proof. Due to Proposition 2.3, since T is known and appears as a factor in the claimed running time, we can assume that $T = 1$ without loss of generality. Also, note that determining n is enough to compute F_{GC}. Indeed, if a process determines n at round t', it can wait until round $\max\{t', 2Tn\}$ and run the algorithm in Theorem 4.1, which is guaranteed to give the correct output by that time.

In order to determine n assuming that $T = 1$, we let each process run the algorithm in Listing 2 with input $(\mathcal{V}, \ell)$, where $\mathcal{V}$ is the view of the process at the current round t'. We will prove that this algorithm returns a positive integer (as opposed to "Unknown") within $(\ell^2 + \ell + 1)n$ rounds, and the returned number is indeed the correct size of the system n.

Algorithm description. Let b be the number of branches in $\mathcal{V}$ representing leader processes (Line 5). The initial goal of the algorithm is to compute $\ell - b + 1$ approximations of n using the information found in as many disjoint intervals $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{\ell-b+1}$ of levels of $\mathcal{V}$ (Lines 6–17).

If there are not enough levels in $\mathcal{V}$ to compute the desired number of approximations, or if the approximations are not all equal, the algorithm returns "Unknown" (Lines 11 and 15).

In order to compute an approximation of n, say in an interval of levels $\mathcal{L}_i$ starting at L_s, the algorithm goes through at most ℓ phases (Lines 8–16). The first phase begins by calling `ApproxCount` with starting level L_s and $x = \ell$, i.e., the maximum possible value for the anonymity of a leader node (Line 9). Specifically, `ApproxCount` chooses a leader node in $\tau \in L_s$ and tries to estimate n using as few levels as possible.

Let (n', t) be the pair of values returned by `ApproxCount`. If $n' = -1$, this is evidence that $\mathcal{V}$ is still missing some relevant nodes, and therefore "Unknown" is immediately returned (Line 11). If $n' = -2$, then a descendant of τ with multiple children in $\mathcal{V}$ was found, say at level L_t, before an approximation of n could be determined. As this is an undesirable event, the algorithm moves $\mathcal{L}_i$ after L_t and tries again to estimate n (Line 12). If $n' = -3$, then x may not be the correct anonymity of the leader node τ (see the description of `ApproxCount`), and therefore the algorithm calls `ApproxCount` again, with the same starting level L_s, but now with $x = \ell - 1$. If $n' = -3$ is returned again, then $x = \ell - 2$ is tried, and so on. After all possible assignments down to $x = 1$

have failed, the algorithm just moves $\mathcal{L}_i$ forward and tries again from $x = \ell$.

As soon as $n' > 0$, this approximation of n is stored in the variable n^*. If it is different from the previous approximations, then "Unknown" is returned (Line 15). Otherwise, the algorithm proceeds with the next approximation in a new interval of levels $\mathcal{L}_{i+1}$, and so on.

Finally, when $\ell - b + 1$ approximations of n (all equal to n^*) have been found, a correctness check is performed: the algorithm takes the last level L_{t^*} visited thus far; if the current round t' satisfies $t' \geq t^* + n^*$, then n^* is accepted as correct; otherwise "Unknown" is returned (Lines 18–20).

Correctness and running time. We will prove that, if the output of Listing 2 is not "Unknown", then it is indeed the number of processes, i.e., $n^* = n$. Since the $\ell - b + 1$ approximations of n have been computed on disjoint intervals of levels, there is at least one such interval, say $\mathcal{L}_j$, where no leader node in the history tree has more than one child (because there can be at most ℓ leader branches). With the notation of Lemma 4.2, this implies that $a(\tau) = a(\tau')$ whenever `ApproxCount` is called in $\mathcal{L}_j$. Also, since the option $x = \ell$ is tried first, the assumption $x \geq a(\tau)$ of Lemma 4.2 is initially satisfied. Note that `ApproxCount` cannot return $n' = -1$ or $n' = -2$, or else $\mathcal{L}_j$ would not yield any approximation of n. Moreover, by statement (ii) and by the terminating condition (Line 19), if $n' > 0$ while $x \geq a(\tau)$, then $n^* = n' = n$. On the other hand, by statement (i), we necessarily have $n' > 0$ by the time $x = a(\tau)$.

It remains to prove that Listing 2 actually gives an output other than "Unknown" within the claimed number of rounds; it suffices to show that it does so if it is executed at round $t' = (\ell^2 + \ell + 1)n$. It is known that all nodes in the first $t' - n = \ell(\ell + 1)n$ levels of the history tree are contained in the view $\mathcal{V}$ at round t' (cf. [24, Corollary 4.3]). Also, it is straightforward to prove by induction that the assumption of statement (iii) of Lemma 4.2 holds every time `ApproxCount` is invoked. Indeed, in any interval of $(\ell + 1)n$ levels, either a branching leader node is found or a new approximation of n is computed. Since there can be at most ℓ leader branches, at least one approximation of n is computed within $\ell(\ell + 1)n$ levels. Because all nodes in these levels must appear in $\mathcal{V}$, the condition $a(\tau) = a(\tau')$ of Lemma 4.2 is satisfied in all intervals $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{\ell-b+1}$. Reasoning as in the previous paragraph, we conclude that all such intervals must yield the correct approximation of n. So, every time Line 15 is executed, we have $n^* = n'$, and the algorithm cannot return "Unknown". □

5 Negative Results

In this section we provide simple proofs of several negative results and counterexamples, some of which are well known (in particular, Proposition 5.1 is implied by [28, Theorem III.1]). The purpose is to justify all of the assumptions made in Sections 3 and 4.

5.1 Leaderless Networks

Proposition 5.1. *No function other than the frequency-based multi-aggregate functions can be computed with no leader, even when restricted to simple connected static networks.*

Proof. Let $m_1, m_2, \ldots, m_k$ be integers greater than 2 with $\gcd(m_1, m_2, \ldots, m_k) = 1$, and let B be the complete k-partite graph with partite sets $V_1, V_2, \ldots, V_k$ of sizes $m_1, m_2, \ldots, m_k$, respectively. For any positive integer α, construct the static network G_α consisting of α disjoint copies of B, augmented with k cycles $C_1, C_2, \ldots, C_k$ such that, for each $1 \leq i \leq k$, the cycle C_i spans all the αm_i processes in the α copies of V_i. Clearly, G_α is a simple connected static network.

Let the function λ_α assign input z_i to all processes in the α copies of V_i in G_α, and let $\lambda = \lambda_1$. As a result, $\mu_{\lambda_\alpha} = \{(z_1, \alpha m_1), (z_2, \alpha m_2), \ldots, (z_k, \alpha m_k)\} = \alpha \cdot \mu_\lambda$ for all $\alpha \geq 1$. Moreover, all the networks G_α have isomorphic history trees. This is because, at every round, each process in any of the copies of V_i receives exactly two messages from other processes in copies of V_i and exactly m_j messages from processes in copies of V_j, for all $j \neq i$. Thus, it can be proved by induction that all processes in the copies of V_i have isomorphic views, regardless of α.

Due to the fundamental theorem of history trees [24, Theorem 3.1], all the processes with input z_i must give the same output $\psi(z_i, \mu_\lambda) = \psi(z_i, \mu_{\lambda_\alpha}) = \psi(z_i, \alpha \cdot \mu_\lambda)$, regardless of α. Hence, by definition, only frequency-based multi-aggregate functions can be computed in these networks. □

Proposition 5.2. *No algorithm can solve the Average Consensus problem in a T-union-connected leaderless network in less than $2Tn - O(T)$ rounds (with or without termination).*

Proof. According to [24, Theorem 5.2], the number of processes n in a network with $\ell = 1$ and $T = 1$ cannot be determined in less than $2n - O(1)$ rounds (with or without termination). We can reduce this problem to Average Consensus with $\ell = 0$ and $T = 1$ as follows. In any given network with $\ell = T = 1$, assign input 1 to the leader and clear its leader flag; assign input 0 to all other processes. If the processes can compute the mean input value, $1/n$, they can invert it to obtain n in the same number of rounds. It follows that Average Consensus with $\ell = 0$ and $T = 1$ cannot be solved in less than $2n - O(1)$ rounds; this immediately generalizes to an arbitrary T by Proposition 2.3. □

Proposition 5.3. *No algorithm can solve the leaderless Average Consensus problem with explicit termination if nothing is known about the size of the network, even when restricted to simple connected static networks.*

Proof. Assume for a contradiction that there is such an algorithm $\mathcal{A}$. Let $\mathcal{G}$ be a static network consisting of three processes forming a cycle, and assign input 0 to all of them. If the processes execute $\mathcal{A}$, they eventually output the mean value 0 and terminate, say in t rounds.

Now construct a static network $\mathcal{G}'$ consisting of a cycle of $2t + 2$ processes $p_1, p_2, \dots, p_{2t+2}$; assign input 1 to p_1 and input 0 to all other processes. It is easy to see that, from round 0 to round t, the view of the process p_{t+1} is isomorphic to the view of any process in $\mathcal{G}$. Therefore, if p_{t+1} executes $\mathcal{A}$, it terminates in t rounds with the incorrect output 0. Thus, $\mathcal{A}$ is incorrect. □

5.2 Networks with Leaders

Proposition 5.4. *No function other than the multi-aggregate functions can be computed (with or without termination), even when restricted to simple connected static networks with a known number of leaders.*

Proof. It is sufficient to construct a static network where all processes with the same input are indistinguishable at every round. Such is, for example, the complete graph K_n, where the output of a process can only depend on its input and the multiset of all processes' inputs [24, Theorem 5.1]. This is the definition of a multi-aggregate function, and thus no other functions can be computed. □

Proposition 5.5. *No algorithm can compute the Counting function F_C (with or without termination) with no knowledge about ℓ, even when restricted to simple connected static networks with a known and arbitrarily small ratio ℓ/n.*

Proof. Let us fix a positive integer k; we will construct an infinite class of networks whose ratio ℓ/n is $1/k$ as follows. For every $i \geq 3$, let $\mathcal{G}_i$ be the static network consisting of a cycle of $n_i = k \cdot i$ processes of which $\ell_i = i$ are leaders, such that the leaders are evenly spaced among the non-leaders. Assume that all processes get the same input (apart from their leader flags). Then, at any round, all the leaders in all of these networks have isomorphic views, which are independent of i. It follows that, if nothing is known about ℓ_i (other than the ratio ℓ_i/n_i, which is fixed), all the leaders in all the networks $\mathcal{G}_i$ always give equal outputs. Since the number of processes n_i depends on i, it follows that at most one of these networks can stabilize on the correct output n_i. ☐

Proposition 5.6. *For any $\ell \geq 1$, no algorithm can compute the Counting function F_C (with or without termination) in all simple T-union-connected networks with ℓ leaders in less than $T(2n-\ell)-O(T)$ rounds.*

Proof. It was shown in [24, Theorem 5.2] that there is a family of simple 1-union-connected networks $\mathcal{G}_n$, with $n \geq 1$, with the following properties. $\mathcal{G}_n$ has $\ell = 1$ leader and n processes in total; moreover, up to round $2n - O(1)$, the leaders of $\mathcal{G}_n$ and $\mathcal{G}_{n+1}$ have isomorphic views.

Let us fix $\ell \geq 1$, and let us construct $\mathcal{G}'_n$, for $n \geq \ell$, by attaching a chain of $\ell - 1$ additional leaders $p_1, p_2, \dots, p_{\ell-1}$ to the single leader p_ℓ of $\mathcal{G}_{n-\ell+1}$ at every round. Note that $\mathcal{G}'_n$ has n processes in total and a stable subpath $(p_1, p_2, \dots, p_\ell)$ which is attached to the rest of the network via p_ℓ.

It is straightforward to see that the process p_ℓ in $\mathcal{G}'_n$ and the process p_ℓ in $\mathcal{G}'_{n+1}$, which correspond to the leaders of $\mathcal{G}_{n-\ell+1}$ and $\mathcal{G}_{n-\ell+2}$ respectively, have isomorphic views up to round $2(n-\ell)-O(1)$. Since the view of p_1 is completely determined by the view of p_ℓ, and it takes $\ell - 1$ rounds for any information to travel from p_ℓ to p_1, we conclude that the process p_1 in $\mathcal{G}'_n$ and the process p_1 in $\mathcal{G}'_{n+1}$ have isomorphic views up to round $2n - \ell - O(1)$.

Thus, up to that round, the two processes must give an equal output, implying that they cannot both output the number of processes in their respective networks. It follows that the Counting function with $\ell \geq 1$ leaders and $T = 1$ cannot be computed in less than $2n - \ell - O(1)$ rounds, which generalizes to an arbitrary T by Proposition 2.3. ☐

6 Conclusions

We have shown that anonymous processes in disconnected dynamic networks can compute all the multi-aggregate functions and no other functions, provided that the network contains a known number of leaders $\ell \geq 1$. If there are no leaders or the number of leaders is unknown, the class of computable functions reduces to the frequency-based multi-aggregate functions. We have also identified the functions F_{GC} and F_R as the complete problems for each class. Notably, the network's dynamic disconnectivity T does not affect the computability of functions, but only makes computation slower.

Moreover, we gave efficient stabilizing and terminating algorithms for computing all the aforementioned functions. Some of our algorithms make assumptions on the processes' a-priori knowledge about the network; we proved that such assumptions are actually necessary. All our algorithms have optimal linear running times in terms of T and the size of the network n.

In one case, there is still a small gap in terms of the number of leaders ℓ. Namely, for terminating computation with $\ell \geq 1$ leaders, we have a lower bound of $T(2n-\ell)-O(T)$ rounds (Proposition 5.6) and an upper bound of $(\ell^2+\ell+1)Tn$ rounds (Theorem 4.3). Although these bounds asymptotically match if the number of leaders ℓ is constant (which is a realistic assumption in most applications), optimizing them with respect to ℓ is left as an open problem.

Observe that our stabilizing algorithms use an unbounded amount of memory, as processes keep adding nodes to their view at every round. This can be avoided if the dynamic disconnectivity T (as well as an upper bound on n, in case of a leaderless network) is known: In this case, processes can run the stabilizing and the terminating version of the relevant algorithm in parallel, and stop adding nodes to their views when the terminating algorithm halts. It is an open problem whether a stabilizing algorithm for F_{GC} or F_R can use a finite amount of memory with no knowledge of T.

Our algorithms require processes to send each other explicit representations of their history trees, which have cubic size in the worst case [24]. It would be interesting to develop algorithms that only send messages of logarithmic size, possibly with a trade-off in terms of running time. We are currently able to do so for leaderless networks and networks with a unique leader, but not for networks with more than one leader [25].

We also wonder if our results hold more generally for networks where communications are not necessarily synchronous. We conjecture that our algorithms can be generalized to networks where messages may be delayed by a bounded number of rounds or processes may be inactive for some rounds (provided that a "global fairness" condition is met).

APPENDIX

A The Subroutine ApproxCount and Its Correctness

In this section we define the subroutine `ApproxCount`$(\mathcal{V}, s, x, \ell)$ introduced in Section 4.2 and invoked in Listing 2. Its starting point is the algorithm in [24, Section 4.2], with the added difficulty that here we have a strand of leader nodes in the view $\mathcal{V}$ hanging from the first leader node τ in level L_s, where the anonymity $a(\tau)$ is an unknown number not greater than ℓ (as opposed to $a(\tau) = 1$, which is assumed in [24]). The algorithm begins by assuming that $a(\tau)$ is the given parameter x, and then it makes deductions on the anonymities of other nodes until it is able to make an estimate $n' > 0$ on the total number of processes, or report failure in the form of an error code $n' \in \{-1, -2, -3\}$. In particular, since the algorithm requires the existence of a long-enough strand hanging from τ, it reports failure if some descendants of τ (in the relevant levels of $\mathcal{V}$) have more than one child. Another important difficulty that is unique to the multi-leader case is that, even if $\mathcal{V}$ contains a long-enough strand of leader nodes, some nodes in the strand may still be branching in the history tree (that is, the chain of leader nodes is branching, but only one branch appears in $\mathcal{V}$). We will first revisit and generalize [24, Section 4.2] in order to formally state our new subroutine and prove its correctness and running time. We will conclude this section with a proof of Lemma 4.2.

We remark that `ApproxCount` assumes that the network is 1-union-connected, as this is sufficient for the main result of Section 4.2 to hold for any T-union-connected network (see the proof of Theorem 4.3).

Discrepancy δ. Suppose that `ApproxCount` is invoked with arguments $\mathcal{V}$, s, x, ℓ, where $1 \le x \le \ell$, and let τ be the first leader node in level L_s of $\mathcal{V}$ (if τ does not exist, the procedure immediately returns the error code $n' = -1$). We define the *discrepancy* δ as the ratio $x/a(\tau)$. Clearly, $1/\ell \le \delta \le \ell$. Note that, since $a(\tau)$ is not a-priori known by the process executing `ApproxCount`, then neither is δ.

Conditional anonymity. `ApproxCount` starts by assuming that the anonymity of τ is x, and makes deductions on other anonymities based on this assumption. Thus, we will distinguish between the actual anonymity of a node $a(v)$ and the *conditional anonymity* $a'(v) = \delta a(v)$ that `ApproxCount` may compute under the initial assumption that $a'(\tau) = x = \delta a(\tau)$.

Guessing conditional anonymities. Let u be a node of a history tree, and assume that the conditional anonymities of all its children u_1, u_2, ..., u_k have been computed: such a node u is called a *guesser*. If v is not among the children of u but it is at their same level, and the red edge $\{v, u\}$ is present with multiplicity $m \ge 1$, we say that v is *guessable* by u. In this case, we can make a *guess* $g(v)$ on the conditional anonymity $a'(v)$:

$$g(v) = \frac{a'(u_1) \cdot m_1 + a'(u_2) \cdot m_2 + \cdots + a'(u_k) \cdot m_k}{m}, \qquad (1)$$

where m_i is the multiplicity of the red edge $\{u_i, v'\}$ for all $1 \le i \le k$, and v' is the parent of v (possibly, $m_i = 0$). Note that $g(v)$ may not be an integer. Although a guess may be inaccurate, it never underestimates the conditional anonymity:

Lemma A.1. *If v is guessable, then $g(v) \ge a'(v)$. Moreover, if v has no siblings, $g(v) = a'(v)$.*

Proof. Let $u, v' \in L_t$, and let P_1 and P_2 be the sets of processes represented by u and v', respectively.

By counting the links between P_1 and P_2 in G_{t+1} in two ways, we have

$$\sum_i a(u_i)\, m_i = \sum_i a(v_i)\, m'_i \geq a(v)\, m,$$

where the two sums range over all children of u and v', respectively (note that $v = v_j$ for some j), and m'_i is the multiplicity of the red edge $\{v_i, u\}$ (so, $m = m'_j$). Our lemma now easily follows from the above equation and from the definition of conditional anonymity. □

Heavy nodes. As the algorithm in [24], also our subroutine `ApproxCount` assigns guesses in a *well-spread* fashion, i.e., in such a way that at most one node per level is assigned a guess.

Suppose now that a node v has been assigned a guess. We define its *weight* $w(v)$ as the number of nodes in the subtree hanging from v that have been assigned a guess (this includes v itself). Recall that subtrees are determined by black edges only. We say that v is *heavy* if $w(v) \geq \lfloor g(v) \rfloor$.

Lemma A.2. *Assume that $\delta \geq 1$. In a well-spread assignment of guesses, if $w(v) > a'(v)$, then some descendants of v are heavy (the* descendants *of v are the nodes in the subtree hanging from v other than v itself).*

Proof. Our proof is by well-founded induction on $w(v)$. Assume for a contradiction that no descendants of v are heavy. Let $v_1, v_2, \ldots, v_k$ be the "immediate" descendants of v that have been assigned guesses. That is, for all $1 \leq i \leq k$, no internal nodes of the black path with endpoints v and v_i have been assigned guesses (observe that $k \geq 1$ because, by assumption, $w(v) > a'(v) = \delta a(v) \geq a(v) \geq 1$).

By the basic properties of history trees, $a(v) \geq \sum_i a(v_i)$, and therefore $a'(v) \geq \sum_i a'(v_i)$. Also, the induction hypothesis implies that $w(v_i) \leq a'(v_i)$ for all $1 \leq i \leq k$, or else one of the v_i's would have a heavy descendant. Therefore,

$$w(v) - 1 = \sum_i w(v_i) \leq \sum_i a'(v_i) \leq a'(v) < w(v).$$

Observe that all the terms in this chain of inequalities are between the two consecutive integers $w(v) - 1$ and $w(v)$. It follows that

$$w(v_i) \leq a'(v_i) < w(v_i) + 1$$

for all $1 \leq i \leq k$. Also,

$$a'(v) - 1 < \sum_i a'(v_i) \leq a'(v).$$

However, since every conditional anonymity is an integer multiple of the discrepancy $\delta \geq 1$, we conclude that $a'(v) = \sum_i a'(v_i)$. Hence, $a(v) = \sum_i a(v_i)$.

Let v_d be the deepest of the v_i's, which is unique, since the assignment of guesses is well spread. Note that v_d has no siblings at all, otherwise we would have $a(v) > \sum_i a(v_i)$. Due to Lemma A.1, we have $g(v_d) = a'(v_d)$. Thus,

$$w(v_d) \leq a'(v_d) = g(v_d) < w(v_d) + 1,$$

which implies that $\lfloor g(v_d) \rfloor = w(v_d)$, and so v_d is heavy. □

Correct guesses. We say that a node v has a *correct* guess if v has been assigned a guess and $g(v) = a'(v)$. The next lemma gives a criterion to determine if a guess is correct.

Lemma A.3. *Assume that $\delta \geq 1$. In a well-spread assignment of guesses, if a node v is heavy and no descendant of v is heavy, then v has a correct guess or the guess on v is not an integer.*

Proof. If $g(v)$ is not an integer, there is nothing to prove. Otherwise, because v is heavy, $g(v) = \lfloor g(v) \rfloor \leq w(v)$. Since v has no heavy descendants, Lemma A.2 implies $w(v) \leq a'(v)$. Also, by Lemma A.1, $a'(v) \leq g(v)$. We conclude that

$$g(v) \leq w(v) \leq a'(v) \leq g(v).$$

Therefore $g(v) = a'(v)$, and v has a correct guess. □

When the criterion in Lemma A.3 applies to a node v, we say that v has been *counted*. So, counted nodes are nodes that have been assigned a guess, which was then confirmed to be the correct conditional anonymity.

Cuts and isles. Fix a view $\mathcal{V}$ of a history tree $\mathcal{H}$. A set of nodes C in $\mathcal{V}$ is said to be a *cut* for a node $v \notin C$ of $\mathcal{V}$ if two conditions hold: (i) for every leaf v' of $\mathcal{V}$ that lies in the subtree hanging from v, the black path from v to v' contains a node of C, and (ii) no proper subset of C satisfies condition (i). A cut for the root r whose nodes are all counted is said to be a *counting cut*.

Let s be a counted node in $\mathcal{V}$, and let F be a cut for v whose nodes are all counted. Then, the set of nodes spanned by the black paths from s to the nodes of F is called *isle*; s is the *root* of the isle, while each node in F is a *leaf* of the isle. The nodes in an isle other than the root and the leaves are called *internal*. An isle is said to be *trivial* if it has no internal nodes.

If s is an isle's root and F is its set of leaves, we have $a(s) \geq \sum_{v \in F} a(v)$, because s may have some descendants in the history tree $\mathcal{H}$ that do not appear in the view $\mathcal{V}$. This is equivalent to $a'(s) \geq \sum_{v \in F} a'(v)$. If equality holds, then the isle is said to be *complete*; in this case, we can easily compute the conditional anonymities of all the internal nodes by adding them up starting from the nodes in F and working our way up to s.

Overview of `ApproxCount`. Our subroutine `ApproxCount` is found in Listing 3. It repeatedly assigns guesses to nodes based on known conditional anonymities, starting from τ and its descendants. Eventually some nodes become heavy, and the criterion in Lemma A.3 causes the deepest of them to become counted. In turn, counted nodes eventually form isles; the internal nodes of complete isles are marked as counted, which gives rise to more guessers, and so on. In the end, if a counting cut is created, the algorithm checks whether the conditional anonymities of the leader nodes in the cut add up to ℓ.

Algorithmic details of `ApproxCount`. The algorithm `ApproxCount` uses flags to mark nodes as "guessed" or "counted"; initially, no node is marked. Thanks to these flags, we can check if a node $u \in \mathcal{V}$ is a guesser: let $u_1, u_2, \ldots, u_k$ be the children of u that are also in $\mathcal{V}$ (recall that a view does not contain all nodes of a history tree); u is a *guesser* if and only if it is marked as counted, all the u_i's are marked as counted, and $a'(u) = \sum_i a'(u_i)$ (which implies $a(u) = \sum_i a(u_i)$, and thus no children of u are missing from $\mathcal{V}$).

`ApproxCount` will ensure that nodes marked as guessed are well-spread at all times; if a level of $\mathcal{V}$ contains a guessed node, it is said to be *locked*. A level L_t is *guessable* if it is not locked and has a non-counted node v that is guessable, i.e., there is a guesser u in L_{t-1} and the red edge $\{v, u\}$ is present in $\mathcal{V}$ with positive multiplicity.

The algorithm starts by assigning a conditional anonymity $a'(\tau) = x$ to the first leader node $\tau \in L_s$. (If no leader node exists in L_s, it immediately returns the error code -1, Line 6.) It also finds the longest strand P_τ hanging from τ, assigns the same conditional anonymity x to all of its nodes (including the unique child of the last node of P_τ) and marks them as counted (Lines 7–11).

Listing 3: The subroutine `ApproxCount` invoked in Listing 2

```
# Input: a view $\mathcal{V}$ and three integers $s$, $x$, $\ell$
# Output: a pair of integers $(n', t)$

Let $L_{-1}$, $L_0$, $L_1$, ... be the levels of $\mathcal{V}$
Assign $t := s$
If $L_s$ does not contain any leader nodes, return $(-1, t)$
Let $\tau$ be the first leader node in $L_s$
Mark all nodes in $\mathcal{V}$ as not guessed and not counted
Assign $u := \tau$; assign $a'(u) := x$; mark $u$ as counted
While $u$ has a unique child $u'$ in $\mathcal{V}$
   Assign $u := u'$; assign $a'(u) := x$; mark $u$ as counted
While there are guessable levels and a counting cut has not been found
   Let $v$ be a guessable non-counted node of smallest depth in $\mathcal{V}$
   Let $L_{t'}$ be the level of $v$; assign $t := \max\{t, t'\}$
   Assign a guess $g(v)$ to $v$ as in Equation (1); mark $v$ as guessed
   Let $P_v$ be the black path from $v$ to its ancestor in $L_s$
   If there is a heavy node in $P_v$
      Let $v'$ be the heavy node in $P_v$ of maximum depth
      If $g(v')$ is not an integer, return $(-3, t)$
      Assign $a'(v') := g(v')$; mark $v'$ as counted and not guessed
      If $v'$ is the root or a leaf of a non-trivial complete isle $I$
         For each internal node $w$ of $I$
            Assign $a'(w) := \sum_{w' \text{ leaf of } I \text{ and descendant of } w} a'(w')$
            Mark $w$ as counted and not guessed
If no counting cut has been found, return $(-2, t)$
Else
   Let $C$ be a counting cut between $L_s$ and $L_t$
   Let $n' = \sum_{v \in C} a'(v)$
   Let $\ell' = \sum_{v \text{ leader node in } C} a'(v)$
   If $\ell' < \ell$, return $(-1, t)$
   If $\ell' > \ell$, return $(-3, t)$
   Return $(n', t)$
```

Then, as long as there are guessable levels and no counting cut has been found yet, the algorithm keeps assigning guesses to non-counted nodes (Line 12).

When a guess is made on a node v, some nodes in the path from v to its ancestor in L_s may become heavy; if so, let v' be the deepest heavy node. If $g(v')$ is not an integer, the algorithm returns the error code -3 (Line 19). (As we will prove later, this can only happen if $\delta \neq 1$ or some nodes in the strand P_τ have children that are not in the view $\mathcal{V}$.) Otherwise, if $g(v')$ is an integer, the algorithm marks v' as counted (Line 20), in accordance with Lemma A.3. Furthermore, if the newly counted node v' is the root or a leaf of a complete isle I, then the conditional anonymities of all the internal nodes of I are determined, and such nodes are marked as counted; this also unlocks their levels if such nodes were marked as guessed (Lines 21–24).

In the end, the algorithm performs a "reality check" and possibly returns an estimate n' of n, as follows. If no counting cut was found, the algorithm returns the error code -2 (Line 25). Otherwise, a counting cut C has been found. The algorithm computes n' (respectively, ℓ') as the sum of the conditional anonymities of all nodes (respectively, all leader nodes) in C. If $\ell' = \ell$, then the algorithm returns n' (Line 32). Otherwise, it returns the error code -1 if $\ell' < \ell$ (Line 30) or the error code -3 if $\ell' > \ell$ (Line 31). In all cases, the algorithm also returns the maximum depth t of a guessed or counted node (excluding τ and its descendants), or s if no such node exists.

Consistency condition. In order for our algorithm to work properly, a condition has to be satisfied whenever a new guess is made. Indeed, note that all of our previous lemmas on guesses rest on the assumption that the conditional anonymities of a guesser and all of its children are known. However, while the node τ has a known conditional anonymity (by definition, $a'(\tau) = x$), the same is not necessarily true of the descendants of τ and all other nodes that are eventually marked as counted by the algorithm. This justifies the following definition.

Condition 1. *During the execution of* `ApproxCount`, *if a guess is made on a node v at level $L_{t'}$ of $\mathcal{V}$, then τ has a (unique) descendant $\tau' \in L_{t'}$ and $a(\tau) = a(\tau')$.*

As we will prove next, as long as Condition 1 is satisfied during the execution of `ApproxCount`, all of the nodes between levels L_s and L_t that are marked as counted do have correct guesses (i.e., their guesses coincide with their conditional anonymities). Note that in general there is no guarantee that Condition 1 will be satisfied at any point; it is the job of our main counting algorithm in Section 4.2 to ensure that the condition is satisfied often enough for our computations to be successful.

Correctness. In order to prove the correctness of `ApproxCount`, it is convenient to show that it also maintains some *invariants*, i.e., properties that are always satisfied as long as some conditions are met.

Lemma A.4. *Assume that $\delta \geq 1$. Then, as long as Condition 1 is satisfied, the following hold.*

(i) The nodes marked as guessed are always well spread.

(ii) Whenever Line 13 is reached, there are no heavy nodes.

(iii) Whenever Line 13 is reached, all complete isles are trivial.

(iv) The conditional anonymity of any node between L_s and L_t that is marked as counted has been correctly computed.

Proof. Statement (i) is true by design with no additional assumptions, because the algorithm always makes a new guess in a guessable level, which is not locked by definition. Thus, no two nodes marked as guessed are ever in the same level, and so they are well spread.

All other statements can be proved collectively by induction. They certainly hold the first time Line 13 is ever reached. Indeed, the only nodes marked as counted up to this point are τ and some of its descendants, which are assigned the conditional anonymity x. Since $s = t$ and τ has conditional anonymity x by definition, statement (iv) is satisfied. Note that some descendants of τ that are marked as counted may not have been assigned their correct conditional anonymities, because some branches of the history tree may not appear in $\mathcal{V}$. However, no guesses have been made yet, and therefore no nodes are heavy; thus, statement (ii) is satisfied. Moreover, the only isles are formed by τ and its descendants, and are obviously all trivial; so, statement (iii) is satisfied.

Now assume that statements (ii), (iii), and (iv) are all satisfied up to some point in the execution of the algorithm. In particular, due to statement (iv), all nodes that have been identified as guessers by the algorithm up to this point were in fact guessers according to our definitions. For this reason, all guesses have been computed as expected, and all of our lemmas on guesses apply (because $\delta \geq 1$).

The next guess on a new node v is performed properly, as well. Indeed, Condition 1 states that τ has a descendant τ' at the same level as v such that $a(\tau') = a(\tau)$, and therefore $a'(\tau') = a'(\tau) = x$; so, τ' has the correct conditional anonymity. Thus, regardless of what the guesser of v is (either the parent of τ' or some other counted node), the guess at Line 15 is computed properly.

Hence, if a node is identified as heavy at Lines 17–18, it is indeed heavy according to our definitions. Because statement (ii) held before making the guess on v, it follows that any heavy node must have been created after the guess, and therefore should be on the path P_v, defined as in Line 16. If no heavy nodes are found on the path, then nothing is done and statements (ii), (iii), and (iv) keep being true.

Otherwise, by Lemma A.3, the deepest heavy node v' on P_v has a correct guess and can be marked as counted, provided that the guess is an integer. Thus, statement (iv) is still true after Line 20. At this point, there are no heavy nodes left, because v' is no longer guessed and all of its ancestors along P_v end up having the same weight they had before the guess on v was made.

Now, because statement (iii) held before marking v' as counted, there can be at most one non-trivial complete isle, and v' must be its root or one of its leaves. Note that, due to statement (iv), any isle I identified as complete at Line 21 is indeed complete according to our definitions. Since I is complete, computing the conditional anonymities of its internal nodes as in Line 23 is correct, and therefore statement (iv) is still true after Line 24. Also, the unique non-trivial isle I gets partitioned into non-trivial isles, and statement (iii) holds again. Finally, since Lines 21–24 may only cause weights to decrease, statement (ii) keeps being true. □

Running time. We will now study the running time of `ApproxCount`. We will prove two lemmas that allow us to give an upper bound on the number of rounds it takes for the algorithm to return an output, provided that some conditions are satisfied.

Lemma A.5. *Assume that $\delta \geq 1$. Then, as long as Condition 1 holds, whenever Line 13 is reached, at most δn levels are locked.*

Proof. Note that the assumptions of Lemma A.4 are satisfied, and therefore all the conditional anonymities and weights assigned to nodes up to this point are correct according to our definitions.

We will begin by proving that, if the subtree hanging from a node v of $\mathcal{V}$ contains more than $a'(v)$ nodes marked as guessed, then it contains a node v' marked as guessed such that $w(v') > a'(v')$. The proof is by well-founded induction based on the subtree relation in $\mathcal{V}$. If v is guessed, then we can take $v' = v$. Otherwise, by the pigeonhole principle, v has at least one child u whose hanging subtree contains more than $a'(u)$ guessed nodes. Thus, v' is found in this subtree by the induction hypothesis.

Assume for a contradiction that more than δn levels of $\mathcal{V}$ are locked; hence, $\mathcal{V}$ contains more than δn nodes marked as guessed. Since the conditional anonymity of the root r of $\mathcal{V}$ is δn, by the above paragraph we know that $\mathcal{V}$ contains a guessed node v' such that $w(v') > a'(v')$. Since $\delta \geq 1$ and Lemma A.4 (i) holds, we can apply Lemma A.2 to v', which implies that there exist heavy nodes. In turn, this contradicts Lemma A.4 (ii). We conclude that at most δn levels are locked. □

We say that a node v of the history tree $\mathcal{H}$ is *missing* from level L_i of the view $\mathcal{V}$ if v is at the level of $\mathcal{H}$ corresponding to L_i but does not appear in $\mathcal{V}$. Clearly, if a level of $\mathcal{V}$ has no missing nodes, all previous levels also have no missing nodes.

Lemma A.6. *Assume that $\delta \geq 1$. Then, as long as level L_t of $\mathcal{V}$ is not missing any nodes (where t is defined and updated as in* `ApproxCount`*), whenever Line 13 is reached, there are at most $n-2$ levels in the range from L_{s+1} to L_t that lack a guessable non-counted node.*

Proof. By definition of t, either $t = s$ or the algorithm has performed at least one guess on a node at level L_t with a guesser at level L_{t-1}. It is easy to prove by induction that the first guesser to perform a guess on this level must be the unique descendant $\tau' \in L_{t-1}$ of the selected leader node $\tau \in L_s$. Moreover, both τ' and its unique child in $\mathcal{V}$ have been assigned conditional anonymity x at Lines 9–11, and the same is true of all nodes in the black path P_τ from τ to τ', which is a strand in $\mathcal{V}$. Since level L_t is not missing any nodes, then each of the nodes in P_τ has a unique child in the history tree, as well. It follows that all descendants of τ up to level L_t have the same anonymity as τ. Also, by definition of t and the way it is updated (Line 14), no guesses have been made on nodes at levels deeper than L_t, and hence Condition 1 is satisfied up to this point. Thus, Lemma A.4 applies.

Observe that there are no counting cuts, or Line 13 would not be reachable. Due to Lines 9–11, all of the nodes in P_τ initially become guessers. Hence, all levels between L_s and L_{t-1} must have a non-empty set of guessers at all times. Consider any level L_i with $s < i \leq t$ such that all the guessable nodes in L_i are already counted. Let S be the set of guessers in L_{i-1}; note that not all nodes in L_{i-1} are guessers, or else they would give rise to a counting cut. Since the network is 1-union-connected, there is a red edge $\{u, v\}$ (with positive multiplicity) such that $u \in S$ and the parent of v is not in S. By definition, the node v is guessable; therefore, it is counted. Also, since the parent of v is not a guesser, v must have a non-counted parent or a non-counted sibling; note that such a non-counted node is in $\mathcal{V}$.

We have proved that every level between L_{s+1} and L_t lacking a guessable non-counted node contains a counted node v having a parent or a sibling that is not counted: we call such a node v a *bad* node. To conclude the proof, it suffices to show that there are at most $n-2$ bad nodes between L_{s+1} and L_t. Observe that no nodes in P_τ can be bad.

We will prove by induction that, if a subtree $\mathcal{W}$ of $\mathcal{V}$ contains the root r, the leader node τ, no counting cuts, and no non-trivial isles, then $\mathcal{W}$ contains at most $f-1$ bad nodes, where f is the number of leaves of $\mathcal{W}$ not in the subtree hanging from τ. The case $f = 0$ is impossible, because the single node τ yields a counting cut. Thus, the base case is $f = 1$, which holds because any bad node v in $\mathcal{W}$ and not in P_τ gives rise to the counting cut $\{\tau, v\}$ (recall that a bad node is counted by definition).

For the induction step, let v be a bad node of maximum depth in $\mathcal{W}$. Let $(v_1, v_2, \dots, v_k)$ be the black path from $v_1 = v$ to the root $v_k = r$, and let $1 < i \leq k$ be the smallest index such that v_i has more than one child in $\mathcal{W}$ (i must exist, because this path eventually joins the black path from τ to r). Let $\mathcal{W}'$ be the tree obtained by deleting the black edge $\{v_{i-1}, v_i\}$ from $\mathcal{W}$, as well as the subtree hanging from it. Notice that the induction hypothesis applies to $\mathcal{W}'$: since v_1 is counted, and each of the nodes $v_2, \dots, v_{i-1}$ has a unique child in $\mathcal{W}$, the removal of $\{v_{i-1}, v_i\}$ does not create counting cuts or non-trivial isles. Also, v_2 is not counted (unless perhaps $v_2 = v_i$), because v_1 is bad. Furthermore, none of the nodes $v_3, \ldots, v_{i-1}$ is counted, or else v_2 would be an internal node of a (non-trivial) isle in $\mathcal{W}$. Therefore, $\mathcal{W}'$ has exactly one less bad node than $\mathcal{W}$ and one less leaf; the induction hypothesis now implies that $\mathcal{W}$ contains at most $f-1$ bad nodes.

Observe that the subtree $\mathcal{V}'$ of $\mathcal{V}$ formed by all levels up to L_t satisfies all of the above conditions, as it contains $\tau \in L_s$, the root r, and has no counting cuts, because a counting cut for $\mathcal{V}'$ would be a counting cut for $\mathcal{V}$, as well (recall that $\mathcal{V}$ has no counting cuts). Also, Lemma A.4 (iii) ensures that $\mathcal{V}'$ contains no non-trivial complete isles. However, since no nodes are missing from the levels of $\mathcal{V}'$, all isles in $\mathcal{V}'$ are complete, and thus must be trivial. We conclude that, if $\mathcal{V}'$ has f leaves not in the subtree hanging from τ, it contains at most $f-1$ bad nodes. Since such leaves induce a partition of the at most $n-1$ processes not represented by τ, we have $f \leq n-1$, implying that the number of bad nodes up to L_t is at most $n-2$. □

Main lemma. We are now ready to prove Lemma 4.2.

Lemma A.7. *Let* `ApproxCount`$(\mathcal{V}, s, x, \ell)$ *return* (n', t). *Assume that* τ *exists and* $x \geq a(\tau)$. *Let* τ' *be the (unique) descendant of* τ *in* $\mathcal{V}$ *at level* L_t, *and let* $L_{t'}$ *be the last level of* $\mathcal{V}$. *Then:*

(i) If $x = a(\tau) = a(\tau')$, *then* $n' \neq -3$.

(ii) If $n' > 0$ *and* $t' \geq t + n'$ *and* $a(\tau) = a(\tau')$, *then* $n' = n$.

(iii) If $t' \geq s + (\ell+2)n - 1$, *then* $s \leq t \leq s + (\ell+1)n - 1$ *and* $n' \neq -1$. *Moreover, if* $n' = -2$, *then* L_t *contains a leader node with at least two children in* $\mathcal{V}$.

Proof. Note that τ' is well defined, because the returned pair is (n', t), which means that either $t = s$, and thus $\tau = \tau'$, or $t > s$, and hence some guesses have been made on level L_t, the first of which must have had the parent of τ' as the guesser.

Let us prove statement (i). The assumption $x = a(\tau)$ implies $\delta = 1$. Moreover, since $a(\tau) = a(\tau')$, Condition 1 is satisfied whenever a guess is made (this is a straightforward induction). Therefore, by Lemma A.4 (iv), all nodes marked as counted up to L_t indeed have the correct guesses. So, the conditional anonymity that is computed for any node is equal to its anonymity ($a'(v) = \delta a(v) = a(v)$), and hence is an integer. This implies that `ApproxCount` cannot return the error code -3 at Line 19. Also, either $\ell' = \ell$ if all leader processes have been counted, or $\ell' < \ell$ if some leader nodes are missing from the view. Either way, `ApproxCount` cannot return the error code -3 at Line 31. We conclude that $n' \neq -3$.

Let us prove statement (ii). Again, because $a(\tau) = a(\tau')$, Condition 1 is satisfied, and all nodes marked as counted have correct guesses. Also, $x \geq a(\tau)$ is equivalent to $\delta \geq 1$. By assumption, `ApproxCount` returns (n', t) with $n' > 0$ and $t' \geq t + n'$. Since $n' > 0$, a counting cut C was found whose nodes are within levels up to L_t, and n' is the sum of the conditional anonymities of all nodes in C. Let S_C be the set of processes represented by the nodes of C; note that $n' \geq |S_C|$, because $\delta \geq 1$. We will prove that S_C includes all processes in the system. Assume the contrary; [24, Lemma 4.2] implies that, since $t' \geq t + n' \geq t + |S_C|$, there is a node $z \in L_t$ representing some process not in S_C. Thus, the black path from z to the root r does not contain any node of C, contradicting the fact that C is a counting cut with no nodes after L_t. Therefore, $|S_C| = n$, i.e., the nodes in C represent all processes in the system. Since `ApproxCount` returns $n' > 0$, the "reality check" $\ell' = \ell$ succeeds (Lines 30–32). However, ℓ' is the sum of the conditional anonymities of all leader nodes in C, and hence $\ell' = \delta\ell$, implying that $\delta = 1$. Thus, $n' = \delta n = n$, as claimed.

Let us prove statement (iii). Once again, $x \geq a(\tau)$ is equivalent to $\delta \geq 1$. By [24, Corollary 4.3], if $L_{t'}$ is the last level of $\mathcal{V}$, then no nodes are missing from level $L_{t'-n}$. In particular, since

$t' - n \leq (\ell+1)n - 1$, no nodes are missing from any level up to $L_{(\ell+1)n-1}$. Let τ'' be the deepest descendant of τ that is marked as counted at Lines 9–11, and let L_p be the level of τ''. By construction, either all children of τ'' are missing from L_{p+1} or at least two children of τ'' are in L_{p+1}. Also note that τ' must be an ancestor of τ'', and so $t \leq p$.

Assume that $p < s + (\ell+1)n - 1$. This implies that no nodes are missing from level L_{p+1}, and therefore τ'' must have at least two children in L_{p+1}. Since $t \leq p$, we have $t < s + (\ell+1)n - 1$, as desired. Now assume that $n' = -2$, which implies that the algorithm was unable to find a counting cut. We claim that in this case $t = p$. So, assume for a contradiction that $t \leq p - 1$. It follows that $\tau' \in L_t$ is a guesser. Recall that L_t and L_{t+1} are not missing any nodes, because $t \leq p$. Since the network is connected at round $t+1$, there is at least one node in L_{t+1} that is guessable by τ', and so L_{t+1} is a guessable level. However, the algorithm cannot return $n' = -2$ as long as there are guessable levels (Line 12). Thus, $t = p$, which means that $\tau' = \tau''$, and hence τ' has at least two children in $\mathcal{V}$, as desired.

Assume now that $p \geq s + (\ell+1)n - 1$, and recall that no nodes are missing from the levels up to $L_{(\ell+1)n-1}$. In particular, since $\delta \leq \ell$, no nodes are missing from the levels in the interval $\mathcal{L}$, which consists of the $(\delta+1)n - 1$ levels from L_{s+1} to $L_{s+(\delta+1)n-1}$. Thus, by definition of p, as long as no guesses are made outside of $\mathcal{L}$, Condition 1 holds, and therefore Lemmas A.5 and A.6 apply. Hence, as long as no guesses are made outside of $\mathcal{L}$, at most δn levels of $\mathcal{L}$ are locked (Lemma A.5) and at most $n-2$ levels of $\mathcal{L}$ lack a guessable non-counted node (Lemma A.6). We conclude that $\mathcal{L}$ always contains at least one guessable level, and no guesses are ever made outside of $\mathcal{L}$ until either a counting cut is found or $n' = -3$ is returned at Line 19. In both cases, $n' = -2$ is not returned, and moreover $t \leq s + (\delta+1)n - 1 \leq s + (\ell+1)n - 1$, as desired.

It remains to prove that $n' \neq -1$. Since τ exists by assumption, the error code -1 cannot be returned at Line 6, and can only be returned at Line 30. In turn, this can only occur if a counting cut has been found and $\ell' < \ell$. However, we have already proved that no level up to L_t is missing any nodes, which implies that the counting cut contains nodes representing all processes, and in particular $\ell' = \delta\ell$. Since $\delta \geq 1$, we have $\ell' \geq \ell$, and the condition at Line 30 is not satisfied. □

B Survey of Related Work

We examine related work on Counting and Average Consensus, distinguishing between the case of dynamic networks with unique IDs, the case of static anonymous networks, and the case of dynamic anonymous networks.

B.1 Dynamic Networks with IDs

The problem of counting the size of a dynamic network has been first studied by the peer-to-peer systems community [39]. In this case having an exact count of the network at a given time is impossible, as processes may join or leave in an unrestricted way. Therefore, their algorithms mainly focus on providing estimates on the network size with some guarantees. The most related is the work that introduced 1-union-connected networks [36]. They show a counting algorithm that terminates in at most $n+1$ rounds when messages are unrestricted and in $O(n^2)$ rounds when the message size is $O(\log n)$ bits. The techniques used heavily rely on the presence of unique IDs and cannot be extended to our settings.

B.2 Anonymous Static Networks

The study of computability on anonymous networks has been pioneered by Angluin in [1] and it has been a fruitful research topics for the last 30 years [1, 7, 12, 13, 14, 27, 49, 52]. A key concept in anonymous networks is the symmetry of the system; informally, it is the indistinguishability of processes that have the same view of the network. As an example, in an anonymous static ring topology, all processes will have the exact same view of the system, and such a view does not change between rings of different size. Therefore, non-trivial computations including counting are impossible on rings, and some symmetry-breaking assumption is needed (such as a leader [27]). The situation changes if we consider topologies that are asymmetric. As an example, on a wheel graph the central process has a view that is unique, and this allows for the election of a leader and the possibility, among other tasks, of counting the size of the network.

Several tools have been developed to characterize what can be computed on a given network topology (examples are views [52] or fibrations [8]). Unfortunately, these techniques are usable only in the static case and are not defined for highly dynamic systems like the ones studied in our work. Regarding the counting problem in anonymous static networks with a leader, [40] gives a counting algorithm that terminates in at most $2n$ rounds.

B.3 Counting in Anonymous Interval-Connected Networks

The papers that studied counting in anonymous dynamic networks can be divided into two periods. A first series of works [11, 20, 21, 40] gave solutions for the counting problem assuming some initial knowledge on the possible degree of a process. As a matter of fact [40] conjectured that some kind of knowledge was necessary to have a terminating counting algorithm. A second series of works [18, 24, 29, 31, 32, 34, 33] has first shown that counting was possible without such knowledge, and then has proposed increasingly faster solutions, culminating with the linear time asymptotically optimal solution of [24]. We remark that all these papers assume that a leader (or multiple leaders in [32]) is present. This assumption is needed to deterministically break the system's symmetry.

Counting with knowledge on the degrees. Counting in interval-connected anonymous networks was first studied in [40], where it is observed that a leader is necessary to solve counting in static (and therefore also dynamic) anonymous networks (this result can be derived from previous works on static networks such as [8, 52]). The paper does not give a counting algorithm but it gives an algorithm that is able to compute an upper bound on the network size. Specifically, [40] proposes an algorithm that, using an upper bound d on the maximum degree that each process will ever have in the network, calculates an upper bound U on the size of the network; this upper bound may be exponential in the actual network size ($U \leq d^n$).

Assuming the knowledge of an upper bound on the degree, [20] given a counting algorithm that computes n. Such an algorithm is really costly in terms of rounds; it has been shown in [11] to be doubly exponential in the network size. The algorithm proposes a mass distribution approach akin to local averaging [51].

An experimental evaluation of the algorithm in [20] can be found in [22]. The result of [20] has been improved in [11], where, again assuming knowledge of an upper bound d on the maximum degree of a process, an algorithm is given that terminates in $O\left(n(2d)^{n+1}\frac{\log n}{\log d}\right)$ rounds. A later paper [21] has shown that counting is possible when each process knows its degree before starting the round (for example, by means of an oracle). In this case, no prior global upper bound on the degree of processes is needed. [21] only show that the algorithm eventually terminates but does not bound the termination time.

We remark that all the above works assume some knowledge on the dynamic network, as an upper bound on the possible degrees, or as a local oracle. Moreover, all of these works give exponential-time algorithms.

Counting without knowledge on the degrees. The first work proposing an algorithm that does not require any knowledge of the network was [19]. The paper proposed an exponential-time algorithm that terminates in $O\left(n^{n+4}\right)$ rounds. Moreover, it also gives an asymptotically optimal algorithm for a particular category of networks (called persistent-distance). In this type of network, a process never changes its distance from the leader.

This result was improved in [29, 31], which presented a polynomial-time counting algorithm. The paper proposes Methodical Counting, an algorithm that counts in $O(n^5 \log^2(n))$ rounds. Similarly to [20, 21], the paper uses a mass-distribution process that is coupled with a refined analysis of convergence time and clever techniques to detect termination. The paper also notes that, using the same algorithm, all algebraic and boolean functions that depend on the initial number of processes in a given state can be computed. In [30, 34], the same authors extended their result to networks where $\ell \geq 1$ leaders are present (with ℓ known in advance), and gave an algorithm that terminates in $O\left(\frac{n^{4+\epsilon}}{\ell}\log^3(n)\right)$ rounds for any $\epsilon > 0$. In particular, when $\ell = 1$, this result improves on the running time of [29, 31]. Improve the running time and deriving tight bounds for counting in the multi-leader case was left as an open problem in [30, 34]. We give a definitive answer to these questions by providing a better running time that is optimal for any constant number of leaders.

Finally, in [32], they show a counting algorithm parameterized by the isoperimetric number of the dynamic network. The technique used is similar to [29, 31], and it uses the knowledge of the isoperimetric number to shorten the termination time. Specifically, for adversarial graphs (i.e., with non-random topology) with ℓ leaders (ℓ is assumed to be known in advance), they give an algorithm terminating in $O\left(\frac{n^{3+\epsilon}}{\ell {i_{min}}^2}\log^3(n)\right)$ rounds, where i_{min} is a known lower bound on the isoperimetric number of the network. This improves the work in [34], but only in graphs where i_{min} is $\omega(1/\sqrt{n})$. The authors also study various types of graphs with stochastic dynamism; we remark that in this case they always obtain superlinear results, as well. The best case is that of Erdős–Rényi, graphs where their algorithm terminates in $O\left(\frac{n^{1+\epsilon}}{\ell {p_{min}}^2}\log^5(n)\right)$ rounds; here p_{min} is the smallest among the probabilities of creating an arc on all rounds. Specifically, if $p_{min} = O(1/n)$, their algorithm is at least cubic.

A recent breakthrough has been shown in [24], which proposed the novel technique of *history trees*. A history tree is a combinatorial structure that models the entire evolution of an anonymous dynamic graph. By developing a theory of history trees for dynamic networks with a unique leader, the authors have shown a terminating solution for the *Generalized Counting* problem in $3n - 3$ rounds.[20] The authors have shown that the Generalized Counting problem is complete for the class of problems solvable in general dynamic networks, proving that computing in anonymous dynamic networks with a unique leader is linear. The authors have also given a stabilizing, non-terminating, algorithm for Generalized Counting that stabilizes in roughly $2n$ rounds, providing an almost matching lower bound (we will discuss lower bounds below). We remark that the results of [24] do not apply to the leaderless or the multi-leader cases; the latter is left as an open problem in [24], which we settle in the present paper.

All the above works assume the dynamic network to be connected at each round. The only work that studied counting in disconnected networks is the recent pre-print [33]. The paper proposes an

[20] In the Generalized Counting problem, each process starts with a certain input, and the problem is solved when each process has computed the multiset of these inputs.

algorithm that solves the Generalized Counting in $\tilde{O}\left((R + \frac{n}{\ell})\frac{n^{2T(1+\epsilon)}}{i_{min}^2}\right)$ rounds,[21] where T is the dynamic disconnectivity, ℓ is the number of leaders, i_{min} is the isoperimetric number of the network, and R is the cumulative bit length of all inputs. In unknown networks, assuming inputs of small size, the number of rounds is roughly $\tilde{O}\left(\frac{n^{2T(1+\epsilon)+3}}{\ell}\right)$. Interestingly, the algorithm requires messages of size $O(\log n)$ bits. We highlight that the algorithm has a running time of roughly $\tilde{O}\left(\frac{n^5}{\ell}\right)$ in dynamic networks that are connected at each round, and the complexity grows exponentially with the dynamic disconnectivity T.

Summarizing, to the best of our knowledge, there exists no worst-case cubic-time algorithm for the Generalized Counting with multiple leaders and no algorithm that scales linearly with the dynamic disconnectivity and the size of the network.

Lower bounds on counting. From [36], a trivial lower bound of $n - 1$ rounds can be derived, as counting obviously requires information from each process to be spread in the network. The first non-trivial lower bound for general dynamic networks has been given in the recent work [24]: any algorithm that stabilizes on the correct count requires $2n - 6$ rounds. We remark that such a lower bound also holds for terminating algorithms. Another interesting lower bound is in [18], which shows a specific category of anonymous dynamic networks with constant temporal diameter (the time needed to spread information from a process to all others is at most 3 rounds), but where counting requires $\Omega(\log n)$ rounds.

B.4 Average Consensus

In the Average Consensus problem, each process v_i starts with an input value $x_i(0)$, and the goal is to compute the average of these initial values. This problem has been studied for decades in the communities of distributed control and distributed computing [6, 15, 16, 17, 32, 42, 45, 47, 51, 54]. In the following, we give an overview that places our result in the current body of knowledge. A more detailed picture can be found in the surveys [26, 43, 46]. We can divide current papers between the ones that give convergent solutions to Average Consensus and the ones that give finite-time solutions.

B.5 Convergent Average Consensus

In convergent Average Consensus algorithms, the consensus is not reached in finite time, but each process has a local value that asymptotically converges to the average. A prototypical family of solutions [6, 51] is based on the so-called *convex combination* algorithms, where each process updates its local value $x_i(r)$ at every round r as follows:

$$x_i(r) = \sum_{\forall v_j \in N(r,v_i)\cup\{v_i\}} a_{ij}(r) \cdot x_j(r-1).$$

The value $a_{ij}(r)$ is taken from a weight matrix that models a dynamic graph. We remark that convex combination algorithms do not need unique IDs, and thus work in anonymous networks.

The ϵ-convergence of an algorithm is defined as the time it takes to be sure that the maximum discrepancy between the local value of a process and the mean is at most ϵ times the initial

[21]A function is $\tilde{O}(f(n))$ if it is $O(f(n)g(n))$ for some polylogarithmic function $g(n)$.

discrepancy. That is, if the mean is $m = \sum_{i \in V} x_i(0)/|V|$, the following should hold:

$$\frac{\max_i \{|x_i(r) - m|\}}{\max_i \{|x_i(0) - m|\}} \leq \epsilon.$$

The local averaging approach has been studied in depth, and several upper and lower bounds for ϵ-convergence are known for both static and dynamic networks [46, 47]. The procedure ϵ-converges in $O\left(Tn^3 \log(\frac{1}{\epsilon})\right)$ rounds if, at every round, the weight matrix $A(r)$ such that $(A(r))_{ij} = a_{ij}(r)$ is doubly stochastic (i.e., the sum of the values on rows and columns is 1) and the dynamic graph is T-interval-connected [42]. In a dynamic network, it is possible to have doubly stochastic weight matrices when an upper bound on the processes' degrees is known [17]. Without such knowledge, if the dynamic graph is always connected and stable (i.e., it changes every two rounds), then it is possible to implement a Metropolis weights strategy that converges in $O\left(n^2 \log(\frac{n}{\epsilon})\right)$ rounds [42].

Numerous other studies have investigated averaging algorithms based on Metropolis rules. However, Metropolis rules require processes to know their out-degree prior to the broadcast phase of each round, making them unsuitable for our model. Charron-Bost et al. observed this in [16]: "Unfortunately, local algorithms cannot implement the Metropolis rule over dynamic networks. The rule is only "local" in the weak sense that an agent's next estimate $x_i(t)$ depends on information present *within distance* 2 of agent i in the communication graph $\mathbb{G}(t)$, which is not local *enough* when the network is subject to change". The only paper that assumes a similar setting to ours is [16], but it restricts the dynamic graph to be 1-interval-connected. The paper shows an algorithm that uses MaxMetropolis weights and converges in $O\left(n^4 \log(\frac{n}{\epsilon})\right)$ rounds.

We remark that all of the above works only achieve convergence to the average of the inputs, and do not stabilize on the average in finite time.

B.6 Finite-Time Average Consensus

The algorithms in the second class solve the finite-time Average Consensus; in this case, the value stabilizes to the actual average in a finite number of rounds. The majority of the literature on finite-time Average Consensus has considered static networks [26, 45, 54]. In such a setting, algorithms that stabilize in a linear number of rounds are known [45, 54]. Few works considered anonymous dynamic networks; [42] describes an algorithm that stabilizes in $O(n^2)$ rounds and requires the network to change every three rounds, while in [15] a randomized Monte Carlo linear algorithm is given. An interesting take is given in [32], which investigates terminating Average Consensus algorithms for adversarial dynamic graphs and random dynamic graphs (i.e., Watts–Strogatz, Barabàsi–Albert, RGG, and Erdős–Rényi–Gilbert graphs). The algorithm for the adversarial case has a time complexity of $O\left(\frac{n^5}{\ell} \log^3(n)\right)$ rounds, where ℓ is the (known) number of leaders in the system. Our multi-leader algorithm is optimal for any constant number of leaders, and settles an open problem of [32] on finding tight results. For random dynamic graphs, the complexity of the algorithm changes according to the model, but all the algorithms presented in [32] are super-linear and require the knowledge of the number of leaders to terminate.

To the best of our knowledge, there is no deterministic solution to Average Consensus that stabilizes, or even converges, in a linear number of rounds in unknown dynamic networks.

References

[1] D. Angluin. Local and Global Properties in Networks of Processors (Extended Abstract). In *Proceedings of the 12th ACM Symposium on Theory of Computing (STOC '80)*, pages 82–93,

1980.

[2] D. Angluin, J. Aspnes, and D. Eisenstat. Fast Computation by Population Protocols with a Leader. *Distributed Computing*, 21(3):61–75, 2008.

[3] J. Aspnes, J. Beauquier, J. Burman, and D. Sohier. Time and Space Optimal Counting in Population Protocols. In *Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS '16)*, pages 13:1–13:17, 2016.

[4] J. Beauquier, J. Burman, S. Clavière, and D. Sohier. Space-Optimal Counting in Population Protocols. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC '15)*, pages 631–646, 2015.

[5] J. Beauquier, J. Burman, and S. Kutten. A Self-stabilizing Transformer for Population Protocols with Covering. *Theoretical Computer Science*, 412(33):4247–4259, 2011.

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., USA, 1989.

[7] P. Boldi and S. Vigna. An Effective Characterization of Computability in Anonymous Networks. In *Proceedings of the 15th International Conference on Distributed Computing (DISC '01)*, pages 33–47, 2001.

[8] P. Boldi and S. Vigna. Fibrations of Graphs. *Discrete Mathematics*, 243:21–66, 2002.

[9] A. Casteigts, F. Flocchini, B. Mans, and N. Santoro. Shortest, Fastest, and Foremost Broadcast in Dynamic Networks. *International Journal of Foundations of Computer Science*, 26(4):499–522, 2015.

[10] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-Varying Graphs and Dynamic Networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[11] M. Chakraborty, A. Milani, and M. A. Mosteiro. A Faster Exact-Counting Protocol for Anonymous Dynamic Networks. *Algorithmica*, 80(11):3023–3049, 2018.

[12] J. Chalopin, S. Das, and N. Santoro. Groupings and Pairings in Anonymous Networks. In *Proceedings of the 20th International Conference on Distributed Computing (DISC '06)*, pages 105–119, 2006.

[13] J. Chalopin, E. Godard, and Y. Métivier. Local Terminations and Distributed Computability in Anonymous Networks. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC '08)*, pages 47–62, 2008.

[14] J. Chalopin, Y. Métivier, and T. Morsellino. Enumeration and Leader Election in Partially Anonymous and Multi-hop Broadcast Networks. *Fundamenta Informaticae*, 120(1):1–27, 2012.

[15] B. Charron-Bost and P. Lambein-Monette. Randomization and Quantization for Average Consensus. In *Proceedings of the 57th IEEE Conference on Decision and Control (CDC '18)*, pages 3716–3721, 2018.

[16] B. Charron-Bost and P. Lambein-Monette. Computing Outside the Box: Average Consensus over Dynamic Networks. In *Proceedings of the 1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND '22)*, pages 10:1–10:16, 2022.

[17] B. Chazelle. The Total s-Energy of a Multiagent System. *SIAM Journal on Control and Optimization*, 49(4):1680–1706, 2011.

[18] G. A. Di Luna and G. Baldoni. Brief Announcement: Investigating the Cost of Anonymity on Dynamic Networks. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC '15)*, pages 339–341, 2015.

[19] G. A. Di Luna and G. Baldoni. Non Trivial Computations in Anonymous Dynamic Networks. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS '15)*, pages 1–16, 2016.

[20] G. A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Conscious and Unconscious Counting on Anonymous Dynamic Networks. In *Proceedings of the 15th International Conference on Distributed Computing and Networking (ICDCN '14)*, pages 257–271, 2014.

[21] G. A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting in Anonymous Dynamic Networks Under Worst-Case Adversary. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS '14)*, pages 338–347, 2014.

[22] G. A. Di Luna, S. Bonomi, I. Chatzigiannakis, and R. Baldoni. Counting in Anonymous Dynamic Networks: An Experimental Perspective. In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS '13)*, pages 139–154, 2013.

[23] G. A. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta. Population Protocols with Faulty Interactions: The Impact of a Leader. *Theoretical Computer Science*, 754:35–49, 2019.

[24] G. A. Di Luna and G. Viglietta. Computing in Anonymous Dynamic Networks Is Linear. In *Proceedings of the 63rd IEEE Symposium on Foundations of Computer Science (FOCS '22)*, pages 1122–1133, 2022.

[25] G. A. Di Luna and G. Viglietta. Brief Announcement: Efficient Computation in Congested Anonymous Dynamic Networks. In *Proceedings of the 42nd ACM Symposium on Principles of Distributed Computing (PODC '23)*, pages 176–179, 2023.

[26] L. Faramondi, R. Setola, and G. Oliva. Performance and Robustness of Discrete and Finite Time Average Consensus Algorithms. *International Journal of Systems Science*, 49(12):2704–2724, 2018.

[27] P. Fraigniaud, A. Pelc, D. Peleg, and S. Pérennes. Assigning Labels in Unknown Anonymous Networks. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC '00)*, pages 101–111, 2000.

[28] J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Distributed Anonymous Discrete Function Computation. *IEEE Transactions on Automatic Control*, 56(10):2276–2289, 2011.

[29] D. R. Kowalski and M. A. Mosteiro. Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP '18)*, pages 156:1–156:14, 2018.

[30] D. R. Kowalski and M. A. Mosteiro. Polynomial Anonymous Dynamic Distributed Computing Without a Unique Leader. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 147:1–147:15, 2019.

[31] D. R. Kowalski and M. A. Mosteiro. Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations. *Journal of the ACM*, 67(2):11:1–11:17, 2020.

[32] D. R. Kowalski and M. A. Mosteiro. Supervised Average Consensus in Anonymous Dynamic Networks. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21)*, pages 307–317, 2021.

[33] D. R. Kowalski and M. A. Mosteiro. Efficient Distributed Computations in Anonymous Dynamic Congested Systems with Opportunistic Connectivity. *arXiv:2202.07167 [cs.DC]*, pages 1–28, 2022.

[34] D. R. Kowalski and M. A. Mosteiro. Polynomial Anonymous Dynamic Distributed Computing Without a Unique Leader. *Journal of Computer and System Sciences*, 123:37–63, 2022.

[35] F. Kuhn, T. Locher, and R. Oshman. Gradient Clock Synchronization in Dynamic Networks. *Theory of Computing Systems*, 49(4):781–816, 2011.

[36] F. Kuhn, N. Lynch, and R. Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*, pages 513–522, 2010.

[37] F. Kuhn, Y. Moses, and R. Oshman. Coordinated Consensus in Dynamic Networks. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC '11)*, pages 1–10, 2011.

[38] F. Kuhn and R. Oshman. Dynamic Networks: Models and Algorithms. *SIGACT News*, 42(1):82–96, 2011.

[39] E. Le Merrer, A.-M. Kermarrec, and L. Massoulie. Peer to Peer Size Estimation in Large and Dynamic Networks: A Comparative Study. In *Proceedings of the 15th IEEE International Conference on High Performance Distributed Computing (HPDC '06)*, pages 7–17, 2006.

[40] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Naming and Counting in Anonymous Unknown Dynamic Networks. In *Proceedings of the 15th International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS '13)*, pages 281–295, 2013.

[41] O. Michail and P. G. Spirakis. Elements of the Theory of Dynamic Networks. *Communications of the ACM*, 61(2):72, 2018.

[42] A. Nedić, A. Olshevsky, A. E. Ozdaglar, and J. N. Tsitsiklis. On Distributed Averaging Algorithms and Quantization Effects. *IEEE Transactions on Automatic Control*, 54(11):2506–2517, 2009.

[43] A. Nedić, A. Olshevsky, and M. G. Rabbat. Network Topology and Communication-Computation Tradeoffs in Decentralized Optimization. *Proceedings of the IEEE*, 106(5):953–976, 2018.

[44] R. O'Dell and R. Wattenhofer. Information Dissemination in Highly Dynamic Graphs. In *Proceedings of the 5th Joint Workshop on Foundations of Mobile Computing (DIALM-POMC '05)*, pages 104–110, 2005.

[45] A. Olshevsky. Linear Time Average Consensus and Distributed Optimization on Fixed Graphs. *SIAM Journal on Control and Optimization*, 55(6):3990–4014, 2017.

[46] A. Olshevsky and J. N. Tsitsiklis. Convergence Speed in Distributed Consensus and Averaging. *SIAM Journal on Control and Optimization*, 48(1):33–55, 2009.

[47] A. Olshevsky and J. N. Tsitsiklis. A Lower Bound for Distributed Averaging Algorithms on the Line Graph. *IEEE Transactions on Automatic Control*, 56(11):2694–2698, 2011.

[48] N. Sakamoto. Comparison of Initial Conditions for Distributed Algorithms on Anonymous Networks. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 173–179, 1999.

[49] J. Seidel, J. Uitto, and R. Wattenhofer. Randomness vs. Time in Anonymous Networks. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC '15)*, pages 263–275, 2015.

[50] T. Sharma and M. Bashir. Use of Apps in the COVID-19 Response and the Loss of Privacy Protection. *Nature Medicine*, 26(8):1165–1167, 2020.

[51] J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1984.

[52] M. Yamashita and T. Kameda. Computing on an Anonymous Network. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing (PODC '88)*, pages 117–130, 1988.

[53] M. Yamashita and T. Kameda. Computing on Anonymous Networks. I. Characterizing the Solvable Cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.

[54] Y. Yuan, G.-B. Stan, L. Shi, M. Barahona, and J. Goncalves. Decentralised Minimum-Time Consensus. *Automatica*, 49(5):1227–1235, 2013.