Exact size counting in uniform population protocols in nearly logarithmic time

David Doty[1]
Department of Computer Science, University of California, Davis
doty@ucdavis.edu

Mahsa Eftekhari[2]
Department of Computer Science, University of California, Davis
mhseftekhari@ucdavis.edu

Othon Michail[3]
Department of Computer Science, University of Liverpool, UK
Othon.Michail@liverpool.ac.uk

Paul G. Spirakis[4]
Department of Computer Science, University of Liverpool, UK and Computer Technology Institute & Press "Diophantus" (CTI), Patras, Greece
P.Spirakis@liverpool.ac.uk

Michail Theofilatos[5]
Department of Computer Science, University of Liverpool, UK
michail.theofilatos@liverpool.ac.uk

Abstract

We study population protocols: networks of anonymous agents whose pairwise interactions are chosen uniformly at random. The *size counting problem* is that of calculating the exact number n of agents in the population, assuming no leader (each agent starts in the same state). We give the first protocol that solves this problem in sublinear time.

The protocol converges in $O(\log n \log\log n)$ time and uses $O(n^{60})$ states ($O(1)+60\log n$ bits of memory per agent) with probability $1-O(\frac{\log\log n}{n})$. The time to converge is also $O(\log n \log\log n)$ in expectation. Crucially, unlike most published protocols with $\omega(1)$ states, our protocol is *uniform*: it uses the same transition algorithm for any population size, so does not need an estimate of the population size to be embedded into the algorithm. A sub-protocol is the first uniform sublinear-time leader election population protocol, taking $O(\log n \log\log n)$ time and $O(n^{18})$ states. The state complexity of both the counting and leader election protocols can be reduced to $O(n^{30})$ and $O(n^9)$ respectively, while increasing the time to $O(\log^2 n)$.

1 Introduction

Population protocols [4] are networks that consist of computational entities called *agents* with no control over the schedule of interactions with other agents. In a population of n agents, repeatedly a random pair of agents is chosen to interact, each observing the state of the other agent before updating its own state.[6] They are an appropriate model for electronic

[1] Supported by NSF grant CCF-1619343.
[2] Supported by NSF grant CCF-1619343.
[3] Supported by EEE/CS initiative NeST.
[4] Supported by EEE/CS initiative NeST.
[5] Supported by EEE/CS initiative NeST, Leverhulme Research Centre for Functional Materials Design.
[6] Using message-passing terminology, each agent sends its entire state of memory as the message.

computing scenarios such as sensor networks and for "fast-mixing" physical systems such as animal populations [33], gene regulatory networks [13], and chemical reactions [29], the latter increasingly regarded as an implementable "programming language" for molecular engineering, due to recent experimental breakthroughs in DNA nanotechnology [15, 31].

The *(parallel) time* for some event to happen in a protocol is a random variable, defined as the number of interactions, divided by n, until the event happens. This measure of time is based on the natural parallel model where each agent participates in $\Theta(1)$ interactions each unit of time; hence $\Theta(n)$ total interactions per unit time [6]. In this paper all references to "time" refer to parallel time. The original model [4] stipulated that each agent is a finite-state machine, with a state set Q constant with respect to the population size, and a transition function $\delta : Q \times Q \to Q \times Q$ indicating that if agents in states $a \in Q$ and $b \in Q$ interact, and $\delta(a, b) = (c, d)$, then they update respectively to states c and d. Many important distributed computing problems provably require $\Omega(n)$ time for population protocols to solve under the constraint of $O(1)$ states, such as leader election [16], exact majority computation [1], and many other predicates and functions [10].

This limitation on time-efficient computation with constant-memory protocols motivates the study of population protocols with memory that can grow with n. A recent blitz of impressive results has shown that leader election [1, 11, 12, 17] and exact majority [2, 3] can be solved in polylog(n) time using polylog(n) states. Notably, each of these protocols requires an approximate estimate of the population size n to be encoded into each agent (commonly a constant-factor upper bound on $\lfloor \log n \rfloor$ or $\lfloor \log \log n \rfloor$).

This *partially* motivates our study of the *size counting problem* of computing the population size n. The problem is clearly solvable by an $O(n)$ time protocol using a straightforward leader election: Agents initially assume they are leaders and the count is 1. When two leaders meet, one agent sums their counts while the other becomes a follower, and followers propagate by epidemic the maximum count. No faster protocol was previously known.

Our study is further motivated by the desire to understand the power of *uniform* computation in population protocols. All of the mentioned positive results with $\omega(1)$ states [1–3, 11, 12, 17, 26, 27] use a *nonuniform* model: given n, the state set Q_n and transition function $\delta_n : Q_n \times Q_n \to Q_n \times Q_n$ are allowed to depend arbitrarily on n, other than the constraint that $|Q_n| \leq f(n)$ for some "small" function f growing as polylog(n).[7] This nonuniformity is used to encode a value such as $\lfloor \log n \rfloor$ into the cited protocols.[8]

We define a stricter *uniform* variant of the model: the same transition algorithm is used for all populations, though the number of states may vary with the population size (formalized with Turing machines; see Section 2.) A uniform protocol can be deployed into *any* population without knowing in advance the size, or even a rough estimate of the size. The original, $O(1)$-state model [4–6], is uniform since there is a single transition function. Because we allow memory to grow with n, our model's power exceeds that of the original, but is strictly less than that of the nonuniform model of most papers using $\omega(1)$ states.

[7] Another constraint, sometimes explicitly stated, e.g., [1], but usually implicit from the constructions, requires that if $|Q_n| = |Q_m|$ for $n \neq m$, then $Q_n = Q_m$ and $\delta_n = \delta_m$.

[8] In these papers [1–3, 11, 12, 17, 26, 27], the role of the value $\log n$ (or $\log \log n$) is as a threshold to compare to some other integer k, which starts at 0 and increments, stopping some stage of the protocol when $k \geq \log n$. A naïve attempt to achieve uniformity is to initialize the comparison threshold to some constant $c < \log n$, which is then updated by the agent with each interaction in such a way that c "quickly" reaches some value $\geq \log n$. The challenge, however, is that prior to the event $c \geq \log n$, the comparison "$k \geq c$?" should never evaluate to true and cause an erroneous early termination of the stage, nor should a fast-growing c "overshoot" $\log n$ and excessively increase the memory requirement.

1.1 Contribution

Our main result is a uniform protocol that, with probability $\geq 1 - \frac{10+5\log\log n}{n}$, counts the population size, converging in $6 \log n \log\log n$ time using $2^{15}n^{60}$ states ($15 + 60 \log n$ bits), without an initial leader (all agents have an initially identical state). The protocol is *stabilizing*: the output is correct with probability 1, converging in expected time $7 \log n \log\log n$.[9]

A key subprotocol performs leader election in time $O(\log n \log\log n)$ with high probability and in expectation. It uses $O(n^{18})$ states, much more than the $O(\log\log n)$ known to be necessary [1] and sufficient [17] for sublinear time leader election. However, it is uniform, unlike all known sublinear-time leader election protocols [1,11,12,17]. It repeatedly increases the length of a binary string each agent stores, where the protocol, with probability at least $1 - O(1/n)$, takes $O(\log n)$ time once the length of this string reaches $\approx \log n$. The length increases in stages that each take $O(\log n)$ time. Our main protocol doubles the string length each stage, so takes $\log\log n$ stages (hence $O(\log n \log\log n)$ time) to reach length $\log n$.

The protocol generalizes straightforwardly to trade off time and memory: by adjusting the rate at which the string length grows, the convergence time $t(n)$ is $O(f(n) \log n)$, where $f(n)$ is the number of stages required for the string length to reach $\log n$. For example, if the code length increments by 1 each stage, then $f(n) = \log n$, so $t(n) = \log^2 n$. In this case the state complexity would be $O(n^{30})$ for the full protocol and $O(n^9)$ for just the leader election portion. (See Section 3.7.) By squaring the string length each stage, $t(n) = \log n \log\log\log n$. By exponentiating the string length, $t(n) = \log n \log^* n$. Even slower-growing $f(n)$ such as inverse Ackermann are achievable. However, the faster the string length grows each stage, the more it potentially overshoots $\log n$, increasing the space requirements. For example, for $t(n) = \log n \log\log\log n$ by repeated squaring, the worst-case string length is $\log^2 n$, meaning $2^{O(\log^2 n)} = n^{O(\log n)}$ states. Multiplying length by a constant gives the fastest increase that maintains a polynomial number of states.

The number of states our protocol uses is very large compared to most population protocol results, which typically have polylog(n) states. However, it is worth noting that a different goalpost is germane for the size counting problem: at least n states are required, since it takes $\log n$ bits merely to write the number n. Our protocol uses a constant factor more bits: about $60 \log n$. Chemical reaction networks are frequently cited as a real system for which population protocols are an appropriate model. It is reasonable to object that since each state corresponds to a different chemical species, such a large number of states is unrealistic. However, biochemistry provides numerous examples of heteropolymers, such as nucleic acids and peptide chains (linear polymers of amino acids that fold into proteins), in which $c = O(1)$ basic monomer types (e.g., the 4 DNA bases A, C, G, T) suffice to construct c^k different polymer types consisting of k monomers. On the engineering side, DNA strand displacement systems [30] can in principle construct and modify such information-rich "combinatorial" polymers in a controllable algorithmic fashion, for example simulating a Turing machine whose length-k tape is represented by $O(k)$ DNA strands [28] or searching for solutions to a quantified Boolean formula [32]. The synthesis cost for such systems would scale with the number of *bits* of memory ($O(1)$ molecules per bit stored), thus only logarithmically with the total number of states. It is thus reasonable to conjecture that reliable algorithmic molecular systems, with moderately sized memories in each molecule, are on the horizon.

[9] The time to reach a stable configuration, from which the output *cannot* change, is $\Omega(n)$ for our protocol. We leave open the question of a protocol that reaches a stable configuration in sublinear time.

1.2 Related Work

For the exact population size counting problem, the most heavily studied case is that of *self-stabilization*, which makes the strong adversarial assumption that arbitrary corruption of memory is possible in any agent at any time, and promises only that eventually it will stop. Thus, the protocol must be designed to work from any possible configuration of the memory of each agent. It can be shown that counting is *impossible* without having one agent (the "base station") that is protected from corruption [9]. In this scenario $\Theta(n \log n)$ time is sufficient [8] and necessary [7] for self-stabilizing counting. Counting has also been studied in the related context of worst-case dynamic networks [14, 19, 21, 22, 24].

In the less restrictive setting in which all nodes start from a pre-determined state, Michail [23] proposed a terminating protocol in which a pre-elected leader equipped with two n-counters computes an approximate count between $n/2$ and n in $O(n \log n)$ parallel time with high probability. Regarding *approximation* rather than exact counting, Alistarh, Aspnes, Eisenstat, Gelashvili, Rivest [1] have shown a uniform protocol that in $O(\log n)$ expected time converges to an approximation n' of the true size n such that with high probability $1/2 \log n \leq \log n' \leq 9 \log n$, i.e., $\sqrt{n} \leq n' \leq n^9$.[10]

Key to our technique is a protocol, due to Mocquard, Anceaume, Aspnes, Busnel, and Sericola [27]. Despite the title of that paper ("Counting with Population Protocols"), it actually solves a different problem, a generalization of the majority problem: count the exact difference between "blue" and "red" agents in the initial population. The protocol assumes an initial leader and that each agent initially stores n exactly. In a follow-up work [26], Mocquard et al. showed a *uniform* protocol that, for any $\epsilon > 0$, computes an approximation of the relative proportion (but not exact number) of "blue" nodes in the population, within multiplicative factor $(1 + \epsilon)$ of the true proportion. The approximation precision ϵ depends on a constant number m, which is encoded in the initial state. They also describe a protocol to find the number of "blue" nodes in the population, However, like [27], this latter protocol is not uniform since the transition function encodes the exact value of n.

In a different network model, Jelasity and Montresor [20] use a similar technique to ours that involves a fast "averaging" similar to [26, 27]. However, they do arbitrary-precision rational number averaging, so have a larger memory requirement (not analyzed). They also assume each agent initially has a unique IDs. Goldwasser, Ostrovsky, Scafuro, Sealfon [18] study a related problem in a synchronous variant of population protocols: assuming that both an adversary and the agents themselves have the ability to create and destroy agents (similar to the more general model of chemical reaction networks), using polylog(n) states, they *maintain* the population size within a multiplicative constant of a target size. This is likely relevant to the exact and approximate size counting problems, since the protocol of [18] must "sense" when the population size is too large or small and react.

2 The model

The system consists of a population of n distributed and anonymous (no unique IDs) *agents*, also called *nodes* or *processes*, that can perform local computations. Each agent is a multitape (r-tape) Turing Machine which is defined by a 6-tuple $M = \langle Q, \Gamma, q_0, \epsilon, F, \delta \rangle$. Q is a

[10] We also require an approximate estimate of n in the subprotocol that computes n exactly, but it is not straightforward to adapt the technique of [1] to our setting. The state complexity would be higher, since our method of estimating n obtains n' such that $n \leq n' \leq n^6$. By squaring n' obtained from [1] to ensure it is at least n, the result could be as large as n^{18}.

finite set of *TM states*, Γ is the binary *tape alphabet* $\{0,1\}$, $q_0 \in Q$ is the *initial TM state*, $F \subseteq Q$ is the set of *final TM states*, and $\delta : Q \times \Gamma^r \to Q \times (\Gamma \times \{L,R,S\})^r$ is the *TM transition function*, where L is left shift, R is right shift and S is no shift. We assume that r is a fixed constant, i.e. independent of the population size.

We define three types of tapes. *Input*, *Output* and *Work* tapes. The Input and Output tapes provide information from and to the other agent during an interaction. The Work tapes are used for storing data and for internal operations, which can be assumed to be additions, subtractions and multiplications (divisions can be performed via the Euclidean Division Algorithm, which divides two integers using additions and subtractions).

Let $r_i < r$ be the number of input tapes, $r_o < r$ the number of output tapes and $r_w < r$ the number of work tapes, where $r_i + r_o + r_w = r$. For any $t \geq 0$ let $I(t), O(t)$ and $W(t)$ be $|V| \times r_i$, $|V| \times r_o$ and $|V| \times r_w$ matrices respectively, such that $I_{v,j}(t)$, $O_{v,j}(t)$ and $W_{v,j}(t)$ are the values of the j-th Input, Output and Work tapes respectively of the agent $v \in V$ at time t. Furthermore, for every $t \geq 0$, let $q(t)$ be a $|V|$-dimensional vector such that $q_v(t)$ is the *state* (or *agent-TM-configuration*) of $v \in V$ at time t. We refer to $q(t)$ as the *configuration* (or *global-configuration*) at time t. We say that a population protocol is *leaderless* if $q_v(0) = q_0 \; \forall \; v \in V$, i.e. all agents have the same state in the initial configuration. We also say that $I(0)$ is the population input at time 0.

Let S be the finite set of binary strings $\{0,1\}^*$. This model is defined on a population V of agents and consists of an *input initialization function* $\iota : S \to S^r \times Q$ and an *output function* $\gamma : S^r \times Q \to D$ (D is the set of output values). Initially, the values of the tapes of each agent are determined by the input initialization function ι, and in every step $t+1 \geq 1$, a pair of agents interacts. During an interaction (a,b) between two agents at time $t+1$, each agent updates its state and copies the contents of its Output tapes to the Input tapes of the other agent ($O_{a,:}(t) \to I_{b,:}(t+1)$ and $O_{b,:}(t) \to I_{a,:}(t+1)$). In addition, they update their states according to the (global) joint transition function $f : Q \times Q \to Q \times Q$ as in standard population protocols.

We furthermore assume that each agent has access to independent uniformly random bits, assumed to be pre-written on a special read-only tape (so that we can use a deterministic TM transition function). This is different from the traditional definition of population protocols, which assumes a deterministic transition function. Several papers [1,11] indicate how to use the randomness built into the interaction scheduler to provide nearly uniform random bits to the agents, using various *synthetic coin* techniques, showing that the deterministic model can effectively simulate the randomized model. In the interest of brevity and simplicity of presentation, we will simply assume in the model that each agent has access to a source of uniformly random bits.

Memory requirements for ExactCounting protocol. In our main protocol, Exact-Counting, the agents need $r_i = r_o = 3$ Input and Output tapes for storing the variables `C`, `LC` and `ave`. The memory requirements (number of bits) are $|\texttt{C}| = 6 \log n$, $|\texttt{LC}| = 12 \log n$ and $|\texttt{ave}| = 18 \log n$. In addition, three Work tapes are needed in order to store the variables `M`, `count` and the constants `isLeader` and `phase` (the first cell of a tape can be used for storing the boolean variable `isLeader`, while `phase` can be stored after that cell). The memory requirements (number of bits) are $|\texttt{M}| = 1 + 18 \log n$, $|\texttt{count}| = 6 \log n$ and $|\texttt{isLeader}| + |\texttt{phase}| = 1 + \log 1184$. Finally, two more Work tapes are needed in order to perform divisions between integers, using the Euclidean Division Algorithm.

Terminology conventions. Throughout this paper, n denotes the number of agents in the population. The *time* until some event is measured as the number of interactions until the event occurs, divided by n, also known as parallel time. This represents a natural model

of time complexity in which we expect each agent to have $O(1)$ interactions per unit of time, hence across the whole population, $\Theta(n)$ total interactions occur per unit time. All references to "time" in this paper refer to parallel time. $\log n$ is the base-2 logarithm of n, and $\ln n$ is the base-e logarithm of n.

For ease of understanding, we will use standard population protocol terminology and not refer explicitly to details of the Turing machine definition except where needed. Therefore a *state* always refers to the TM initial configuration of an agent (leaving out TM state and tape head positions since these are identical in all initial configurations), a *configuration* $\vec{c}$ refers to the length-n vector giving the state of each agent, and *transition function* refers to the function computing the next state of an agent, taking as input its state and the other agent's state, by running its Turing machine until it halts. An *epidemic* [6] is a subprotocol of the form $\delta(i, u) = (i, i)$ starting with one agent with i ("infected") and all other $n - 1$ agents with u ("uninfected"), which in $O(\log n)$ expected time converts all agents to i.

2.1 Stabilization and convergence

A protocol *converges* when it reaches a configuration where all agents have the same output, which does not subsequently change. In our main protocol, agents have a field `count`, and convergence to the correct output occurs when each agent has written the value n into `count` for the last time. Configuration $\vec{c}$ is *stable* if every agent agrees on the output, and no configuration reachable from $\vec{c}$ has a different output in any agent. A protocol *stabilizes* if, with probability 1, it eventually reaches a correct stable configuration. Using this terminology, a protocol *stably solves the exact size counting problem* if, for all $n \in \mathbb{Z}^+$, with probability 1, on a population of n agents, the protocol converges to output n and enters a stable configuration.

If the number of configurations reachable from the initial configuration is finite, then stabilization is equivalent to requiring that, for every configuration $\vec{c}$ reachable from the initial configuration, a correct stable configuration is reachable from $\vec{c}$. It is also equivalent to saying that every fair execution reaches a correct stable configuration, where an execution is *fair* if every configuration that is infinitely often reachable in it is infinitely often reached. Although our protocol as defined has an infinite number of reachable configurations, this is done solely to make the analysis simpler, and it can easily be modified to be finite (see explanation of the UniqueID protocol in Section 3.1).

3 Exact Population Size Counting

This section is devoted to proving the main theorem of our paper:

▶ **Theorem 3.1.** *There is a leaderless, uniform population protocol that stably solves the exact size counting problem. With probability at least* $1 - \frac{10+5\log\log n}{n}$*, the convergence time is at most* $6 \ln n \log\log n$*, and each agent is uses* $15 + 60 \log n$ *bits of memory. The expected time to convergence is at most* $7 \ln n \log\log n$*.*

The stabilization time can be much larger, up to $O(n)$. (See Section 3.6.) Theorem 3.1 follows from Theorems 3.14 and 3.15, which respectively cover the "with high probability" and "stabilization and expected time" parts of Theorem 3.1.

The protocol is ExactCounting. There are four main subprotocols: UniqueID, ElectLeader, Averaging, and Timer, each discussed in detail in later subsections. ExactCounting runs in parallel on all agents, but within an agent, each subprotocol runs

Protocol 1 ExactCounting(rec, sen)

▷ state: strings C (code), LC (leader code), Bool isLeader, ints M, ave, count, phase
▷ initial state of agent: C = LC = ε, isLeader = True, M = ave = count = phase = 1
UniqueID(rec, sen)
ElectLeader(rec, sen)
if rec.LC = sen.LC **then** ▷ separate restarts under different leaders
  Averaging(rec, sen)
  Timer(rec, sen)

sequentially (for correctness each subprotocol must run in the given order). Most state updates use one-way rules for selected agents sen (*sender*) and rec (*receiver*). The only rule that is not one-way is Averaging, in which both sender and receiver update their state. In all other cases, only the receiver potentially updates the state.

High-level overview of ExactCounting protocol. UniqueID eventually assigns to every agent a unique id, represented as a binary string called a *code* C. UniqueID requires $\Omega(n)$ time to converge, but it does not need to converge before it can be used by the other subprotocols. In fact, in other subprotocols, agents do not use each others' codes directly. Agents also have a longer code called a *leader code* LC, such that $2|\mathtt{C}| = |\mathtt{LC}|$ and, for any candidate leader, C is a prefix of LC. ElectLeader elects a leader by selecting the agent whose leader code is lexicographically largest. The code length $|\mathtt{C}|$ will eventually be at least length $\log n$, so $|\mathtt{C}|$ can be used to estimate an upper bound M on the value $3n^3$ to within a polynomial factor. Averaging uses M in a leader-driven protocol that counts the population size exactly, which is correct so long as $\mathtt{M} \geq 3n^3$, by using a fast averaging protocol similar to the one studied by Mocquard et al. [27]. Averaging must be restarted by the upstream UniqueID subprotocol many times, and in fact will be restarted beyond the $O(\log n \log\log n)$ time bound we seek. However, within $O(\log n \log\log n)$ time, Averaging will converge to the correct population size. Subsequent restarts of Averaging will reconverge to the correct output, but prior to convergence will have an incorrect output. Timer is used to detect when Averaging has likely converged, waiting to write output into the count field of the agent. This ensures that after the correct value is written, on subsequent restarts of Averaging, the incorrect values that exist before Averaging re-converges will not overwrite the correct value recorded during the earlier restart.

3.1 UniqueID

We assume that two subroutines are available: For $x, y \in \{0,1\}^*$, Append(x, y) returns xy, and for $m \in \mathbb{Z}^+$, RandBits(m) returns a random string in $\{0,1\}^m$.

Subprotocol 2 UniqueID(rec, sen)

if $|\mathtt{rec.C}| < |\mathtt{sen.C}|$ **then** ▷ If receiver's code shorter than sender's, make same length.
  ExtendCode(rec, $|\mathtt{sen.C}| - |\mathtt{rec.C}|$))
if rec.C = sen.C **then** ▷ If codes are the same, double the length.
  ExtendCode(rec, max(1, $|\mathtt{rec}.C|$))

UniqueID can be viewed as traversing a labeled binary tree, until all agents reach a node unoccupied by any other agent. We say the *level* is the maximum depth (longest code length) of any agent in the population. Initiating a new level happens when two agents with

Subroutine 3 ExtendCode(rec, numBits)

```
if rec.isLeader then                    ▷ extend LC by twice numBits; take new C bits from LC
    newLC ← Append(rec.LC, RandBits(2 · numBits))
    SetNewLeaderCode(rec, newLC)                    ▷ described in Subsection 3.2
    rec.C ← Append(rec.C, newLC[(|rec.C| + 1) .. (|rec.C| + numBits)])
else
    rec.C ← Append(rec.C, RandBits(numBits))
```

Subroutine 4 SetNewLeaderCode(rec, newLC)

```
rec.LC ← newLC
▷ restart Timer and Average protocols whenever LC changes.
rec.phase ← 1
rec.M ← 3 · 2^{3|rec.C|}
if rec.isLeader then
    rec.ave ← rec.M
else
    rec.ave ← 0
```

the same code interact. The receiver doubles the length of its code with uniformly random bits, going twice as deep in the tree. To ensure each agent reaches the new level quickly, agents at deeper levels recruit other agents to that level by epidemic, which generate random bits to reach the same code length.

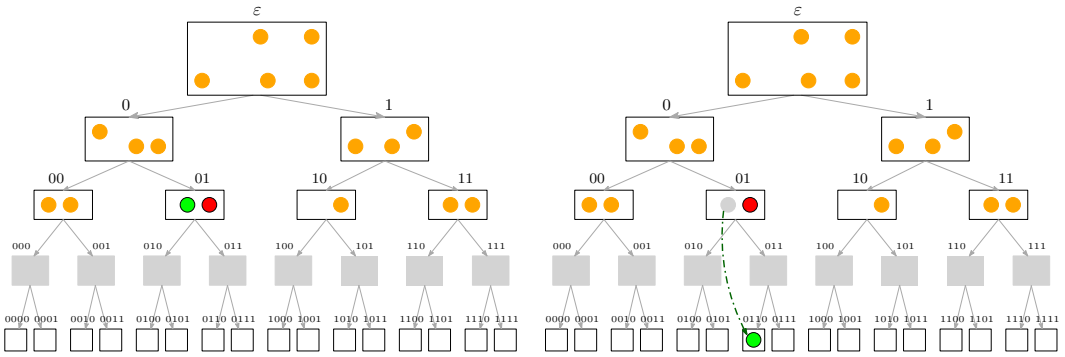
The key property of this protocol is that, in any level $\ell < \log n$, only $O(\log n)$ time is required to increase the level. Since we double the level when it increases, $\log \log n$ such doublings are required to reach level $\geq \log n$, so $O(\log n \log \log n)$ time. Lemma 3.3 formalizes this claim, explaining how the length-increasing schedule can be adjusted to achieve a trade-off between time and memory.

Number of reachable configurations. Since all codes are generated randomly, the number of reachable configurations is infinite. This choice is merely to simplify analysis, allowing us to assume that all agents at a level have uniformly random codes. However, if a finite number of reachable configurations is desired (so that, for instance, the definition of stabilization we use is equivalent to definitions based on reachability), it is possible to modify UniqueID so that when two agents with the same code meet, they *both* append bits that are guaranteed to be different. The protocol still works in this case and in fact takes strictly less expected time for the codes to become unique. Viewing two agents with compatible codes (i.e., one code is a prefix of the other) as equivalent, each new level increases the number of equivalence classes by 1. Thus it is guaranteed that all agents will converge on unique codes of length at most $n - 1$, implying the reachable configuration space is finite.

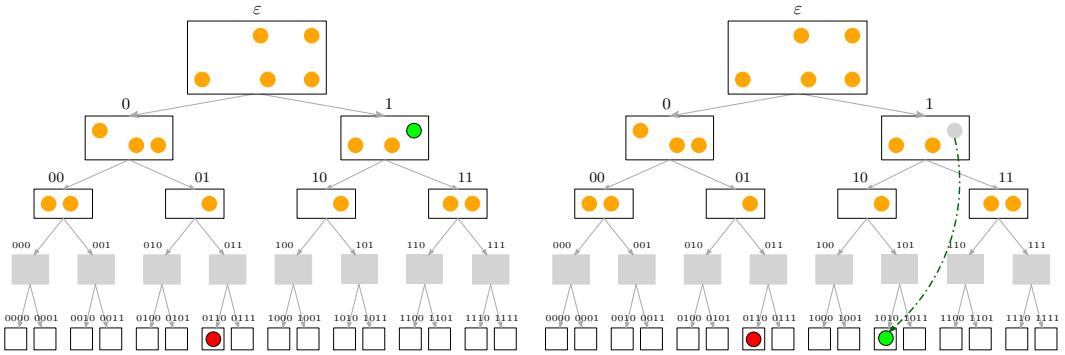
The following lemma is essentially Lemmas 1 and 2 from the paper [6]. However, that paper does not state how the various constants are related, which we require for our proofs. We recapitulate their proof, deriving those relationships explicitly.

▶ **Lemma 3.2** ([6]). *Let T denote the time to complete an epidemic. Then* $\mathrm{E}[T] \leq 4 \ln n$, $\Pr[T < \frac{1}{4} \ln n] < 2e^{-\sqrt{n}}$, *and for any* $\alpha_u > 0$, $\Pr[T > \alpha_u \ln n] < 4n^{-\alpha_u/4+1}$.

Proof. We begin by showing $\Pr[T > \alpha_u \ln n] < 4n^{-\alpha_u/4+1}$. Suppose we have $\alpha_u n \ln n$ interactions, starting with one infected agent. We want to bound the probability that any agent remains uninfected. The second half of an epidemic (after exactly $n/2$ agents



(a) When two agents in the same node at level i interact, receiver moves to a random descendant at level $2i$.



(b) When sender is in a deeper level of the tree, receiver moves to a random descendant in its own subtree at the sender's level.

Figure 1 Agents moving through the binary tree (i.e., choosing binary codes) in accordance with the UniqueID subprotocol.

are infected) has equivalent distribution to the first, so we analyze just the second half, bounding the probability it requires more than $(\alpha_u/2)n \ln n$ interactions. When half of the agents are infected, each interaction picks an infected sender with probability at least $1/2$. The number of interactions to complete the epidemic is then stochastically dominated by a binomial random variable $\mathcal{B}((\alpha_u/2)n \ln n, 1/2)$, equal to the number of heads after $(\alpha_u/2)n \ln n$ coin flips if $\Pr[\text{heads}] = 1/2$.

Let $\mu = E[\mathcal{B}((\alpha_u/2)n \ln n, 1/2)] = (\alpha_u/4)n \ln n$ and $\delta = 2/\sqrt{n}$. By the Chernoff bound [25, Corollary 4.10],

$$\Pr\left[\mathcal{B}((\alpha_u/2)n \ln n, 1/2) < (1-\delta)\mu\right] \quad < \quad e^{-\delta^2 \mu} = e^{-(4/n)(\alpha_u/4)n \ln n} = e^{-\alpha_u \ln n} = n^{-\alpha_u}.$$

So with probability at least $1 - n^{-\alpha_u}$, more than $((1-\delta)\alpha_u/4)n \ln n > (\alpha_u/8)n \ln n$ (since $\delta < 1/2$) interactions involve an infected sender.

To complete the proof of the time upper bound, we need to bound the probability that these $(\alpha_u/8)n \ln n$ interactions fail to infect all agents. Conditioned on each interaction having an infected sender, the random variable giving the number of interactions until all agents are infected is equivalent to the number of collections required to collect the last $n/2$ coupons out of n total. Angluin et al. [6, Lemma 1] showed that for any β, it takes more than $\beta(n/2)\ln(n/2) < (\beta/2)n \ln n$ collections to collect n coupons with probability at most $n^{-\beta+1}$. Let $\beta = \alpha_u/4$. Then $\Pr[(\alpha_u/8)n \ln n$ interactions fail to infect every agent$] \leq n^{-\beta+1} < n^{-\alpha_u/4+1}$. By the union bound on the events *"fewer than* $(\alpha_u/8)n \ln n$ *interactions*

involved an infected sender" and *"$(\alpha_u/8)n \ln n$ interactions fail to infect every agent"*, the second half of the epidemic fails to complete within $(\alpha_u/2)n \ln n$ interactions with probability at most $n^{-\alpha_u} + n^{-\alpha_u/4+1} < 2n^{-\alpha_u/4+1}$.

Again by the union bound on the events *"first half of the epidemic takes more than $(\alpha_u/2)n \ln n$ interactions"* and *"second half of the epidemic takes more than $(\alpha_u/2)n \ln n$ interactions"*, the whole epidemic takes more than $\alpha_u n \ln n$ interactions with probability at most $4n^{-\alpha_u/4+1}$.

To show $\Pr[T < \frac{1}{4} \ln n] < 2e^{-\sqrt{n}}$, we note that Lemma 1 of [6] shows that if S_n is the number of times a coupon must be collected to collect all coupons, then $\Pr[S_n < \frac{1}{4} n \ln n] < 2e^{-\sqrt{n}}$. The proof says $2e^{-\Theta(\sqrt{n})}$, but inspection of the argument reveals that the big-Θ constant can be assumed to be 1. In this case, applying the coupon collector argument to the epidemic, since we are proving a time lower bound, if we assume that *every* interaction involves an infected sender, this process stochastically dominates the real epidemic. Thus $\Pr[T < \frac{1}{4} \ln n] < 2e^{-\sqrt{n}}$.

To analyze the expected time, observe that when k agents are infected, the probability that the next interaction infects an uninfected agent is $\frac{k(n-k)}{n(n-1)} > \frac{k(n-k)}{n^2}$, so expected interactions until an infection at most $\frac{n^2}{k(n-k)}$. By linearity of expectation, the expected number of interactions to complete the epidemic is

$$\begin{aligned}
\sum_{k=1}^{n-1} \frac{n^2}{k(n-k)} &= 2n^2 \sum_{k=1}^{n/2} \frac{1}{k(n-k)} && \text{sum is symmetric about middle index} \\
&< 2n^2 \sum_{k=1}^{n/2} \frac{1}{kn/2} = 4n \sum_{k=1}^{n/2} \frac{1}{k} < 4n \ln(n/2+1) < 4n \ln n,
\end{aligned}$$

i.e., expected time $< 4 \ln n$. ◀

The next lemma bounds the time for UniqueID to reach level at least $\log n$, assuming a generalized way of increasing the level, defining $f(n)$ to be the number of times the level must increase before reaching at least level $\log n$. Afterwards we state a corollary for our protocol, which doubles the level whenever it increases, so $f(n) = \log \log n$. By using this lemma with different choices of f, one can obtain a tradeoff between time and space; if the level increases more (corresponding to a slower-growing f) this takes less time to reach level at least $\log n$, but may overshoot $\log n$ and use more space.

Intuitively, the lemma is proven by observing that the worst case is that the current level is $\log(n) - 1$. It takes $O(\log n)$ time for all agents not yet at that level to reach it by epidemic. At that point the worst case is that codes are distributed to maximize expected time: exactly $n/2$ codes each shared by two agents. Then the expected time is constant for the first interaction between two such agents, starting the next level. Thus it takes time $O(\log n)$ to increase the level, hence $O(f(n) \log n)$ time for the level to increase from 0 to at least $\log n$.

▶ **Lemma 3.3.** *For all $n \in \mathbb{N}$, define $f(n)$ to be the number of times* UniqueID *must increase the level (last line of* UniqueID*) to reach level at least $\log n$. For all $\alpha > 0$, in time $5\alpha f(n) \ln n$, all agents reach level at least $\log n$ with probability at least $1 - 5f(n)n^{-\alpha}$.*

Proof. Imagine an alternate process where at each level agents wait until all other agents also reach the same level before enabling transitions that start the next level (where two agents with the same code meet and the receiver will double its code length). The time for such a process stochastically dominates the time for our protocol, so we can use its

time as an upper bound for our protocol. It suffices to show that, when all agents are at the same level, it takes constant time to start the next level. After initializing a level, by Lemma 3.2, the new code length will spread by epidemic in time $\alpha_u \ln n$ with probability at least $1 - 4n^{-4\alpha_u+1}$.

Assume all agents are currently at level i. Denote by S_j the number of agents in node j of the tree at level i (i.e., they have the j'th code in $\{0,1\}^i$ in lexicographic order). The probability that the next interaction is between two agents at the same node (having equal codes) is minimized when $S_j = S_{j'} = n/2^i$ for all $1 \le j, j' \le 2^i$. Then for all $0 \le i < \log n$, if the current level is i,

$$\begin{aligned}
\Pr\left[\text{next interaction initializes new level}\right] &= \frac{\sum_{j=1}^{2^i}\binom{S_j}{2}}{\binom{n}{2}} = \frac{\sum_{j=1}^{2^i}\binom{n/2^i}{2}}{\binom{n}{2}} \\
&= \frac{2^i(n/2^i)(n/2^i-1)}{n(n-1)} = \frac{n/2^i-1}{n-1} \\
&\ge \frac{n/2^{\log(n)-1}-1}{n-1} = \frac{1}{n-1} > \frac{1}{n}.
\end{aligned}$$

Therefore, the expected number of interactions to start a new level is $\le n$, equivalently parallel time 1. This is a geometric random variable with success probability at least $\frac{1}{n}$. Then at any level $i < \log n$, for any $\alpha'_u > 0$,

$$\begin{aligned}
\Pr\left[\text{initializing next level take more than } \alpha'_u n \ln n \text{ interactions}\right] &= \left(1-\frac{1}{n}\right)^{\alpha'_u n \ln n} \\
&< e^{-\alpha'_u \ln n} = n^{-\alpha'_u}.
\end{aligned}$$

By Lemma 3.2, for any $\alpha_u > 0$, more than $\alpha_u \ln n$ time is required for all agents to reach this level by epidemic with probability at most $4n^{-\alpha_u/4+1}$. By the union bound over this event and the event *"once all agents are at a level, it takes more than time $\alpha'_u \ln n$ to start a new level"* (shown above to happen with probability at most $n^{-\alpha'_u}$), the time spent at each level is more than $\alpha_u \ln n + \alpha'_u \ln n = (\alpha_u + \alpha'_u)\ln n$ with probability at most $4n^{-\alpha_u/4+1} + n^{-\alpha'_u}$. Given $\alpha > 0$, let $\alpha'_u = \alpha$ and $\alpha_u = 4(\alpha+1)$. Then this probability bound is $4n^{-\alpha_u/4+1} + n^{-\alpha'_u} = 5n^{-\alpha}$.

By the union bound over all $f(n)$ levels visited in the tree, it takes more than time $f(n)(\alpha \ln n + 4\alpha \ln n) = 5\alpha f(n) \ln n$ time to reach level at least $\log n$ with probability at most $5f(n)n^{-\alpha}$. ◀

The next corollary is specific to our level-doubling schedule, used throughout the rest of the paper, corresponding to $f(n) = \log\log n$ in Lemma 3.3.

▶ **Corollary 3.4.** *In the* UniqueID *protocol, for all $\alpha > 0$, in $5\alpha \ln n \log\log n$ time all agents reach level at least $\log n$ with probability at least $1 - \frac{5\log\log n}{n^\alpha}$.*

The previous results show UniqueID quickly gets to level $\log n$. The next lemma states that it does not go too far past $\log n$. Intuitively, if the level is $2\log n$, there are n^2 possible codes chosen uniformly at random among n agents, a standard birthday problem with probability $\frac{1}{e}$ of a collision, which drops off polynomially with the level beyond $2\log n$.

▶ **Lemma 3.5.** *Let $\epsilon > 0$. If the current level is $(2+\epsilon)\log n$, all codes are unique with probability at least $1 - \frac{1}{n^\epsilon}$.*

Proof. Consider the agents in order for agent $1, 2, \ldots$. The code of agent i collides with the code of some agent $1, 2, \ldots, i-1$ with probability $\frac{i-1}{c}$, where $c = 2^{(2+\epsilon)\log n} = n^{2+\epsilon}$ is the number of available codes. Then by the union bound,

$$\Pr[\text{at least one collision}] \leq \sum_{i=1}^{n} \frac{i-1}{c} = \frac{n(n-1)}{2c} < \frac{n^2}{n^{2+\epsilon}} = \frac{1}{n^\epsilon}. \quad \blacktriangleleft$$

However, since the code length doubles when it changes, not all values of ϵ in Lemma 3.5 correspond to a level actually visited. It could overshoot by factor two, giving the following.

▶ **Corollary 3.6.** *Let $\epsilon > 0$. The eventual code length of each agent is $< (4 + 2\epsilon)\log n$ with probability at least $1 - \frac{1}{n^\epsilon}$.*

Proof. Recall that a new level of the tree is initiated when two agents with the same code interact. Since the level is doubled in this case, the code lengths exceed $(4 + 2\epsilon)\log n$ if there was a duplicate code at the power-of-two level k such that $(2 + \epsilon)\log n \leq k < (4 + 2\epsilon)\log n \leq 2k$. Over all k satisfying this inequality, the probability of a duplicate code is largest if $k = (2 + \epsilon)\log n$. Applying Lemma 3.5 gives the stated probability bound. ◀

3.2 ElectLeader

Subprotocol 5 ElectLeader(rec, sen)

$p \leftarrow \min(|\text{rec.LC}|, |\text{sen.LC}|)$
if rec.LC$[1..p]$ lexicographically precedes sen.LC$[1..p]$ **then**
▷ Propagate by epidemic the lexicographically greatest leader code
  rec.isLeader $\leftarrow$ False
  SetNewLeaderCode(rec, sen.LC)
if (not rec.isLeader) and ($|\text{rec.LC}| < |\text{sen.LC}|$) **then**
▷ Ensure all leader codes eventually have equal length
  SetNewLeaderCode(rec, sen.LC)

ElectLeader works by propagating by epidemic the "winning" leader code, where a candidate leader drops out if they see an agent (whether leader or follower) with a leader code that beats its own. The trick is to define "win". We compare the shorter leader code with the same-length prefix of the other. If they disagree, the lexicographically largest wins. To ensure all leader code lengths are eventually equal, a follower with the shorter leader code replaces it with the longer one.[11]

The next lemma shows that the leader is probably unique when the population reaches level at least $\log n$. Let $k \in \mathbb{N}$ be such that $\log n \leq k$. When the candidate leaders generate new values of LC upon reaching level k, $|\text{LC}| = 2k \geq 2\log n$. Since there are at least n^2 strings of length $2k \geq 2\log n$, in the worst case, even if all n agents remain candidate leaders at that time, the probability that the lexicographically greatest leader code is duplicated is at most $\frac{1}{n}$.[12] Thus, with probability at least $1 - \frac{1}{n}$, one unique leader has the maximum

[11] Leaders with shorter codes do not replace with longer codes, because it may be that after adding new random bits to get to the current population level, that agent would have the lexicographically largest leader code. This is because a leader with a shorter leader code LC also has a shorter code C, so will eventually catch up in leader code length through the UniqueID protocol.

[12] This almost looks like a birthday problem, but we don't need all the remaining candidate leaders to have a unique leader code, only that the *largest* code appears only once.

leader code, and in $O(\log n)$ time this leader code reaches the remaining candidate leaders by epidemic, who drop out.

▶ **Lemma 3.7.** *At any level $\geq \log n$, with probability $\geq 1 - \frac{1}{n}$, there is a unique leader.*

Proof. Every remaining candidate leader at level $\geq \log n$ has a leader code with at least $2 \log n$ bits. We say i and j *collide* if agents i and j are candidate leaders with the same leader code. Let $X_{i,j}$ be the indicator variable:

$$X_{i,j} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ collide} \\ 0 & \text{otherwise} \end{cases}$$

Let $c \geq 2^{2 \log n} = n^2$ be the number of possible leader codes. Note $\Pr[X_{i,j} = 1] = \frac{1}{c}$. Let $X_i = \sum_{j \neq i} X_{i,j}$; by linearity of expectation $\mathrm{E}[X_i] = \frac{n-1}{c}$. Since $c > n^2$, $\mathrm{E}[X_i] < \frac{n-1}{n^2} < \frac{1}{n}$. By Markov's inequality, $\Pr[X_i \geq 1] \leq \frac{1}{n}$, and the event $X_i \geq 1$ is equivalent to the event that the leader code of agent i is not unique. If we set i to be the agent with the lexicographically greatest leader code of any remaining candidate leader, we conclude that leader is unique with probability $\geq 1 - \frac{1}{n}$. ◀

By Corollary 3.4, the protocol reaches level $\log n$ in $O(\log n \log \log n)$ time. Thus, by Lemma 3.7, with high probability, in time $O(\log n \log \log n)$ ElectLeader converges (the second-to-last candidate leader is eliminated). Unfortunately ElectLeader is not terminating: the remaining leader does not know when it becomes unique. Thus it is not straightforward to compose it with the downstream protocols Averaging and Timer. Standard techniques for making the protocol terminating with high probability, such as setting a timer for a termination signal that probably goes off only after $K \log n \log \log n$ time for a large constant K, do not apply here, because when we start we don't know the value $\log n \log \log n$. Thus, it is necessary, each time a leader adds to its code length, to restart the downstream protocols the existence of a unique leader.[13] This is done in SetNewLeaderCode, which is actually called by both ElectLeader *and* UniqueID, since extending $\mathtt{C}$ for a leader also requires extending $\mathtt{LC}$, to maintain that $2|\mathtt{C}| = |\mathtt{LC}|$.

3.3 Averaging

Subprotocol 6 Averaging(rec, sen)

$$\mathrm{rec.ave}, \mathrm{sen.ave} \leftarrow \left\lceil \frac{\mathrm{rec.ave} + \mathrm{sen.ave}}{2} \right\rceil, \left\lfloor \frac{\mathrm{rec.ave} + \mathrm{sen.ave}}{2} \right\rfloor$$

The previous subsections described how to set up a protocol (perhaps restarted many times) to elect a leader and to produce a value $\mathtt{M} \geq 3n^3$. (With high probability we also have $\mathtt{M} \leq 3 \cdot n^{18}$.) Thus we assume the initial configuration of this protocol is one leader and $n - 1$ followers, each storing this value $\mathtt{M}$, and that the goal is for all of them to converge to a value in $\mathtt{ave}$ such that $n = \left\lfloor \frac{\mathtt{M}}{\mathtt{ave}} + \frac{1}{2} \right\rfloor$.

There is an existing *nonuniform* protocol [27] that can do the following in $O(\log n)$ time. Each agent starts with a bit $b \in \{0, 1\}$ and a number $\mathtt{M} = \Omega(n^{3/2})$. Let n_b be the (unknown)

[13] One might imagine restarts could be tied to the elimination of candidate leaders, which stops within $O(\log n \log \log n)$ time, rather than the extending of codes, which persists for $\Omega(n)$ time. However, the leader may become unique *before* level $\log n$, when $|\mathtt{C}| < \log n$, so $\mathtt{M} = 3 \cdot 2^{2|\mathtt{C}|} < 3 \cdot n^3$ is not sufficiently large to ensure correctness and speed of Averaging. (See Lemma 3.8, which is applied with $c = 1$.)

number of agents storing bit b, so that $n_0 + n_1 = n$. The agents converge to a state in which they all report the value $n_1 - n_0$, the initial different in counts between the two bits.

Their protocol requires that $\mathtt{M} \geq \left\lfloor n^{3/2}/\sqrt{2\delta} \right\rfloor$ to obtain an error probability of $\leq \delta$. The protocol is elegantly simple: agents with $b = 0$ start with an integer value $-\mathtt{M}$, while agents with $b = 1$ start with an integer value $\mathtt{M}$, and state space integers in the interval $\{-\mathtt{M}, -\mathtt{M}+1, \ldots, \mathtt{M}-1, \mathtt{M}\}$. When two agents meet, they average their values, with one taking a floor and the other a ceiling in case the sum of the values is odd. If an agent holds value x, that agent's output is reported as $\lfloor nx/\mathtt{M} + 1/2 \rfloor$, i.e., $nx/\mathtt{M}$ rounded to the nearest integer. This eventually converges to all agents sharing the population-wide average $(n_1 - n_0)\frac{\mathtt{M}}{n}$, and the estimates of this average get close enough for the output to be correct within $O(\log n)$ time [27].

Our protocol essentially inverts this, starting with a known $n_0 = 1$ and $n_1 = n - 1$, computing the population size as a function of the average. The leader starts with value $\mathtt{ave} = \mathtt{M}$, and followers start with $\mathtt{ave} = 0$, and the state space is $\{0, 1, \ldots, \mathtt{M}\}$. The population-wide sum is always $\mathtt{M}$.[14] Eventually all agents have $\mathtt{ave} = \lceil \frac{\mathtt{M}}{n} \rceil$ or $\lfloor \frac{\mathtt{M}}{n} \rfloor$, which could take linear time in the worst case. We show below that with probability at least $1 - n^{-c}$, in $O(\log n)$ time, all agents' ave values are within n^c of $\frac{\mathtt{M}}{n}$. Each agent reports the population size as $\left\lfloor \frac{\mathtt{M}}{\mathtt{ave}} + \frac{1}{2} \right\rfloor$. This is the exact population size n as long as $\mathtt{M} \geq 3n^{c+2}$ and $\mathtt{ave}$ is within n^c of $\frac{\mathtt{M}}{n}$, as the following lemma shows.

▶ **Lemma 3.8.** *Let $c \geq 0$. If $M \geq 3n^{c+2}$, and $x \in \left[\frac{M}{n} - n^c, \frac{M}{n} + n^c\right]$, then $\left\lfloor \frac{M}{x} + \frac{1}{2} \right\rfloor = n$.*

Proof. Since $\left\lfloor \frac{M}{x} + \frac{1}{2} \right\rfloor$ is monotone in x, it suffices to show this holds for the two endpoints of the interval. For the case $x = \frac{M}{n} - n^c$, since $x < \frac{M}{n}$, we have $n < \frac{M}{x}$, and

$$\begin{aligned}
\frac{M}{x} &= \frac{M}{\frac{M}{n} - n^c} = \frac{M}{\frac{M - n^{c+1}}{n}} = \frac{Mn}{M - n^{c+1}} \\
&\leq \frac{Mn}{M - M/(3n)} \quad \text{since } M \geq 3n^{c+2} \\
&= \frac{n}{1 - 1/(3n)} = \frac{n}{(3n-1)/(3n)} = \frac{3n^2}{3n-1} = n + \frac{1}{3(3n-1)} + \frac{1}{3} < n + \frac{1}{2}.
\end{aligned}$$

So $n < \frac{M}{x} < n + \frac{1}{2}$, so $\left\lfloor \frac{M}{x} + \frac{1}{2} \right\rfloor = n$. In the case $x = \frac{M}{n} + n^c$, a similar argument shows that $n - \frac{1}{2} < x < n$. ◀

The above results show that the count computed by AVERAGING is correct if $\mathtt{M}$ is sufficiently large and $\mathtt{ave}$ is within a certain range of the true population-wide average $\frac{\mathtt{M}}{n}$. The next lemma, adapted from [27, Corollary 8], shows that each agent's $\mathtt{ave}$ estimate quickly gets within that range. That corollary is stated in terms of a general upper bound K on how far each agent's $\mathtt{ave}$ field starts from the true population-wide average. In our case, this is given by the leader, which starts with $\mathtt{ave} = \mathtt{M}$, while the true average is $\frac{\mathtt{M}}{n}$, so we choose $K = \mathtt{M} > \mathtt{M} - \frac{\mathtt{M}}{n}$ in Corollary 8 of [27], giving the following.

▶ **Lemma 3.9** ([27]). *For all $\delta \in (0, 1)$ and all $t \geq \ln(4M^2)$, with probability at least $1 - \delta$, after time t, each agent's $\mathtt{ave}$ field is in the interval $\left[\frac{M}{n} - \sqrt{\frac{n}{2\delta}}, \frac{M}{n} + \sqrt{\frac{n}{2\delta}}\right]$.*

▶ **Corollary 3.10.** *Let $c > 0$ and let $\delta = \frac{1}{2n^{2c-1}}$. For all $t \geq \ln(4M^2)$, with probability at least $1 - \delta$, within time t, each agent's $\mathtt{ave}$ field is in the interval $\left[\frac{M}{n} - n^c, \frac{M}{n} + n^c\right]$.*

[14] Think of the leader starting with $\mathtt{M}$ "balls". Interacting agents exchange balls until they have an equal number, or within 1.

Setting $c = 1$ (so $\delta = \frac{1}{2n}$) gives the following corollary.

▶ **Corollary 3.11.** *For all $t \geq \ln(4M^2)$, with probability at least $1 - \frac{1}{2n}$, within time t, each agent's* `ave` *field is in the interval $\left[\frac{M}{n} - n, \frac{M}{n} + n\right]$.*

3.4 Timer

Note that Averaging does not actually write the value $\left\lfloor \frac{\texttt{M}}{\texttt{ave}} + \frac{1}{2} \right\rfloor$ into the `count` field; that is the job of the Timer protocol, which we now explain. The leader is guaranteed with high probability to become unique at least by level $\log n$ (Lemma 3.7). However, since UniqueID likely continues after this point, although the leader is unique, when its level increases, the leader will again generate more bits for its leader code, updating its value `M`, initiating a restart of Averaging. The problem is that although we can prove that the agents likely reach level $\log n$ in $O(\log n \log\log n)$ time, it may take much longer to reach subsequent levels. Thus, although the value `M` estimated at any level $k \geq \log n$ is large enough for Averaging to be correct, if Averaging were to blindly write $\left\lfloor \frac{\texttt{M}}{\texttt{ave}} + \frac{1}{2} \right\rfloor$ into `count` each time `ave` changes, the output will be disrupted while this restart of Averaging converges.

Subprotocol 7 Timer(rec, sen)

▷ run phase clock until `MaxPhase` = 1184 is reached
if rec.`isLeader` and (rec.`phase` = sen.`phase`) and (rec.`phase` < `MaxPhase`) **then**
  rec.`phase` ← rec.`phase` + 1
if (not rec.`isLeader`) and (rec.`phase` < sen.`phase`) **then**
  rec.`phase` ← sen.`phase`
newCount ← ⌊rec.`M`/rec.`ave` + 1/2⌋ ▷ `M`/`ave` rounded to the nearest integer
▷ only write output if timer is done and new count is different
if (rec.`phase` = `MaxPhase`) and (rec.`count` ≠ newCount) and (`M` ≥ 3 · newCount^3) **then**
  rec.`count` ← newCount

We deal with this problem in the following way. When the leader restarts Averaging, it simultaneously restarts Timer, which is a *phase clock* as described by Angluin et al. [6]. Timer is so named because we can find $\beta_l < \beta_u$ and `MaxPhase` such that `MaxPhase` phases of the phase clock will take time between $\beta_l \ln n$ and $\beta_u \ln n$ with high probability. So long as $\beta_l \ln n$ is greater than a high-probability upper bound on the running time of Averaging, the timer likely will not go off (reach the final phase `MaxPhase`) until Averaging has converged. It is only once Timer has reached phase `MaxPhase` that `count` is written, and then only if the new calculated size differs from the previous value in `count`.

There is one additional check done before writing to `count`: if newCount $= \left\lfloor \frac{\texttt{M}}{\texttt{ave}} + \frac{1}{2} \right\rfloor$, we must have `M` $\geq 3 \cdot$ newCount^3 in order to write to `count`. In particular, if `M` $\geq 3n^3$, then newCount cannot be n unless `M` $\geq 3 \cdot$ newCount^3. This is an optimization to save space. Averaging is only guaranteed to get the correct size n efficiently if `M` $\geq 3n^3$. However, when `ave` is small before convergence (e.g., 1) then $\left\lfloor \frac{\texttt{M}}{\texttt{ave}} + \frac{1}{2} \right\rfloor$ can be as large as `M`, requiring $18 \log n$ bits. But if $\left\lfloor \frac{\texttt{M}}{\texttt{ave}} + \frac{1}{2} \right\rfloor = n$ (i.e., is correct) then this value requires at most $\log n$ bits. Since `M` could be as large as $3 \cdot n^{18}$, requiring $O(1) + 18 \log n$ bits, this implies `count` could be as large as n^6, requiring $6 \log n$ bits.

3.5 ExactCounting is fast and correct with high probability

The following is adapted from [6, Corollary 1]. It relates the number of phases in a phase clock to upper and lower bounds on the likely time spent getting to that phase. Our

proof appeals entirely to Corollary 1 of [6] but, unlike [6], the exact relationship between the constants is given in the lemma statement.

▶ **Lemma 3.12** ([6]). *Let $\beta_l, \epsilon_l, \epsilon_u > 0$, and define $p = \max(8\epsilon_l, 32\beta_l)$ and $\beta_u = 4p(\epsilon_u + 2)$. Let T_p be the time needed for a phase clock with $\geq p$ phases to reach phase p. Then for all sufficiently large n, $\Pr[T_p < \beta_l \ln n] < \frac{1}{n^{\epsilon_l}}$ and $\Pr[T_p > \beta_u \ln n] < \frac{1}{n^{\epsilon_u}}$.*

Proof. Based on [6, Corollary 1], setting (variables of [6, Corollary 1] on the left, and our variables on the right) $c = \epsilon_l, d = \beta_l, k = p$, and $a = 1/16$, then by choosing $p = \max(8\epsilon_l, 32 \cdot \beta_l)$, we have $\Pr[T_p < \beta_l \ln n] < \frac{1}{n^{\epsilon_l}}$. By Lemma 3.2, for all $\alpha_u > 0$, the epidemic corresponding to each phase i will complete (all agents reach phase i) in time $> \alpha_u \ln n$ with probability $< 4n^{-\alpha_u/4+1}$. Since the time to complete the epidemic is an upper bound on the time for the leader to interact with an agent in phase i (which could happen before the epidemic completes), we also have that the phase takes time $> \alpha_u \ln n$ with probability $< 4n^{-\alpha_u/4+1}$. By the union bound over all p phases, there exists a phase $1 \leq i \leq p$ taking time $> \alpha_u \ln n$ with probability $< 4pn^{-\alpha_u/4+1}$. Since at least one phase must exceed time $\alpha_u \ln n$ for the sum to exceed $p\alpha_u \ln n$, $\Pr[T_p > p\alpha_u \ln n] < 4pn^{-\alpha_u/4+1}$. Let $\alpha_u = 4(\epsilon_u + 2)$. Substituting $\beta_u = p\alpha_u = 4p(\epsilon_u + 2)$ gives $\Pr[T_p > \beta_u \ln n] < 4pn^{-(4(\epsilon_u+2))/4+1} = 4pn^{-\epsilon_u - 1} < n^{-\epsilon_u}$, which completes the proof. ◀

The next lemma says that Averaging and Timer "happen the way we expect": first Averaging converges, before Timer ends and records the output of Averaging, all in $O(\log n)$ time. When we say "Averaging converges", this refers to the Averaging protocol running in isolation, not as part of a larger protocol that might restart it. That is to say, it may be that Averaging converges, but ExactCounting has not converged, since ExactCounting then restarts Averaging and subsequently changes the `count` field. Intuitively, it follows by a simply union bound on the probability that Averaging is too slow (Corollary 3.11) or Timer is too fast (Lemma 3.12).

▶ **Lemma 3.13.** *For any level $\geq \log n$, if it takes $\geq 14208 \ln n$ time to start the next level, with probability $\geq 1 - \frac{3}{n}$, first* Averaging *converges to the correct output, then* Timer *ends and writes n into* `count`*, in $\leq 14208 \ln n$ time.*

Proof. Corollary 3.6 applied with $\epsilon = 1$ shows that agents codes' length are $\leq 6 \log n$ with probability $\geq 1 - \frac{1}{n}$. Lemma 3.7 shows that after level ℓ the leader is unique with probability $\geq 1 - \frac{1}{n}$. Since $\ell \geq \log n$ the value of M is will be $\geq 3n^3$. By Lemma 3.8, if Averaging converges, then $\left\lfloor \frac{\mathtt{M}}{\mathtt{ave}} + \frac{1}{2} \right\rfloor$ exactly n.

By the union bound on Lemma 3.12 and Corollary 3.11, with probability $\leq \frac{1}{n} + \frac{1}{n}$ the timer takes more than $14208 \ln n$ time, or Averaging takes more than $37 \ln n$ time to converge. Negating these conditions gives conclusion of the lemma.

To show that Timer does not end until Averaging converges, we apply Lemma 3.12 again, but using the time lower bound for Timer. Letting $\epsilon_l = 1$ and $\beta_l = 37$, Lemma 3.12 gives that for $p = 32\beta_l = 1184$, with probability $\geq 1 - \frac{1}{n}$, Timer does not end before $\beta_l \ln n$ time. Applying the union bound to this case and the previous two cases then gives probability $1 - \frac{1}{n}$ as desired. ◀

Finally, we can prove the "with high probability" portion of the main theorem.

▶ **Theorem 3.14.** *With probability at least $1 - \frac{10+5 \log \log n}{n}$,* ExactCounting *converges to the correct output within $6 \ln n \log \log n$ time, and each agent uses at most $15 + 60 \log n$ bits.*

Proof sketch. We sketch the ideas while omitting exact bounds on time and probability. Statements below are "with high probability". Define k to be the unique power of two such that $\log n \leq k < 2\log n$. By Corollary 3.4, level k is reached quickly, so it suffices to prove fast convergence after the event that k is reached. By Lemma 3.7, the leader is unique at level k, therefore also $2k$ and $4k$, and since $k \geq \log n$, $\texttt{M} \geq 3n^3$. So by Lemma 3.13, Averaging will converge within time $t = 14208 \ln n$.

We look at three subcases: among levels $k, 2k, 4k$, one is the earliest among the three where $> t$ time is spent. By Lemma 3.5, level $4k$ is not exceeded since codes are unique. Since codes are unique at level $4k$, at $> t$ (in fact, infinite time) will be spent at level $4k$. But it could be that the protocol also spends time $> t$ at level k or $2k$. Whichever is the first among these three to spend time $> t$, since the previous spent less time, it takes time $\leq 2t$ to reach the first level taking time $> t$. By Lemma 3.13, Averaging converges in time $\leq t$, and by the time *upper* bound of Lemma 3.12, Timer reaches phase `MaxPhase` in time $\leq t$ and records the output of Averaging.

If we are at level k or $2k$, then we might go to a new level. If this is guaranteed to happen within $O(\log n)$ time, then we would not need the Timer protocol. We could simply claim that Averaging will converge at the last level reached, whether k, $2k$, or $4k$. The problem that Timer solves is that ExactCounting may reach level k quickly, Averaging converges quickly, yet a small number of duplicate nodes remain, say 2. It takes $\Omega(n)$ time for them to interact and increase to level $2k$, which restarts Averaging. By the time *lower* bound of Lemma 3.12, in each of these restarts Timer will not reach phase `MaxPhase` until Averaging reconverges, so the `count` field will not be overwritten. Thus convergence happened at the *first* level where we spent time $> t$, even if there are subsequent restarts. ◀

Proof. By Corollary 3.4, agents reach level $\log n$ before $5 \ln n \log\log n$ time with probability at least $1 - \frac{5\log\log n}{n}$. By Lemma 3.7, the leader is unique at any such level, with probability at least $1 - \frac{1}{n}$. Also $\texttt{M} = 3 \cdot 2^{3|C|} \geq 3 \cdot 2^{3\log n} = 3n^2$. Hence, if we could stop UniqueID at this point, then based on Lemma 3.13, Averaging would converge in at most $14208 \ln n$ time with probability at least $1 - \frac{3}{n}$. However, there may be duplicate codes, so UniqueID possibly continues, restarting Averaging later.

Let $t = 14208 \ln n$. For any ℓ, define Round_ℓ to be the event that ExactCounting spends more than time t at level ℓ. Define k to be the unique power of two such that $\log n \leq k < 2\log n$. We consider the following disjoint cases that cover all possible outcomes:

Round_k: Lemma 3.13 shows that with probability at least $1 - \frac{3}{n}$ Averaging converges to the correct output and this output is recorded. By the time *upper* bound of Lemma 3.12, with probability at least Timer reaches phase `MaxPhase` in time $\leq t$ and records the output of Averaging. It remains to show that convergence is likely; i.e., this correct value will not be overwritten. Again using Lemma 3.13, with probability at most $\frac{3}{n}$, at level $2k$ Timer ends before Averaging, and similarly for error probability $\frac{3}{n}$ at level $4k$. By the union bound over these three subcases, in this case, with probability $\geq 1 - \frac{3+3+3}{n} = 1 - \frac{9}{n}$, ExactCounting converges in time $< t$.

(not Round_k) and Round_{2k}: Similar to above, we apply Lemma 3.13 to level $2k$ to obtain probability $\geq 1 - \frac{3}{n}$ that Averaging converges and Timer writes n into `count` in time $< t$, and apply Lemma 3.13 to level $4k$ to obtain probability probability at most $\frac{3}{n}$ that Timer ends too early and disrupts convergence. By the union bound over these two subcases, in this case, with probability $\geq 1 - \frac{3+3}{n} = 1 - \frac{6}{n}$, ExactCounting converges in time $< t$.

(not Round$_k$) and (not Round$_{2k}$) and Round$_{4k}$: Apply Lemma 3.13 to level $4k$ to obtain probability $\geq 1 - \frac{3}{n}$ that AVERAGING converges and TIMER writes n into `count` in time $< t$.

not Round$_{4k}$: Lemma 3.5 applied with $\epsilon = 2$ gives probability at most $\frac{1}{n^2}$ that there are duplicate codes and we reach subsequent levels.

Since the four cases are disjoint, take the maximum error probability of any of them: with probability at least $1 - \frac{9}{n}$, once at level k, it takes time $< t$ to converge.

By the union bound on the event that it takes more than time $5 \ln n \log\log n$ to reach level $\log n$ (probability $\leq \frac{5 \log\log n}{n}$), the event that the leader is not unique (probability $\leq \frac{1}{n}$), and the event that once at level k, it takes time $\geq t$ to converge (probability $\leq \frac{9}{n}$), we obtain that with probability at least $1 - \frac{10+5\log\log n}{n}$, EXACTCOUNTING converges in time $< 6 \ln n \log\log n$.

We now prove the memory requirements. First, if the maximum level reached is ℓ, then the memory requirements are ℓ for `C`, 2ℓ for `LC`, $2 + 3\ell$ for `M`, $2 + 3\ell$ for `ave`, ℓ for `count`, 1 for `isLeader`, and for `phase`, $\log \texttt{MaxPhase} = \log 1184 < 12$, summing to $17 + 10\ell$.

There are two disjoint cases: $2k \geq 3 \log n$ and $2k < 3 \log n$. In the former case, applying Lemma 3.5 with $\epsilon = 1$ gives probability at most $\frac{1}{n}$ of a duplicate code. In the latter case, $4 \log n \leq 4k < 6 \log n$, and applying Lemma 3.5 with $\epsilon = 2$ gives probability at most $\frac{1}{n^2}$ of a duplicate code. In either case the level is less than $6 \log n$, so we set $\ell = 6 \log n$. The sum of the bit requirements is then $15 + 60 \log n$ as needed. Since the cases are disjoint, we take the maximum error probability $\frac{1}{n}$.

Taking a union bound between this event of "too much memory" and the previous event of "too much time", the total error probability bound is $\frac{10+5\log\log n}{n}$ as required. ◀

3.6 ExactCounting converges in fast expected time

Most of the technical difficulty of our analysis is captured by the "with high probability" results stated already. EXACTCOUNTING is also stabilizing, meaning that with probability 1 it gets to a correct configuration that is *stable* (the output cannot change). Probability 1 correctness is required for the expected correct convergence time to be finite, and indeed it asymptotically matches the high probability convergence time of $O(\log n \log\log n)$. However, the protocol takes longer to stabilize, up to $O(n)$ time, since it does not stabilize until UNIQUEID stabilizes.[15]

The next theorem shows a fast expected convergence time, and it completes the second portion of the main result, Theorem 3.1.

▶ **Theorem 3.15.** EXACTCOUNTING *converges in expected time* $7 \ln n \log\log n$.

First we establish some other claims necessary to prove Theorem 3.15. Recall that a protocol *stabilizes* if it converges to the correct output with probability 1.

▶ **Lemma 3.16.** EXACTCOUNTING *stabilizes to the correct population size.*

Proof. Since each agent generates code bits uniformly at random, any pair of agents has probability 0 to generate the same infinite sequence of bits. So with probability 1 all

[15] Prior to that, it is possible, with low probability, after n is written into each agent's `count` field, for a subsequent restart of AVERAGING to write incorrect values, if the corresponding restart of TIMER completes too quickly. So no configuration is stable until UNIQUEID converges and triggers the final restart.

agents eventually have unique codes, and UniqueID stabilizes. We now show that implies ElectLeader stabilizes.

Since there are n agents, UniqueID cannot terminate until at least level $\log n$. So when UniqueID terminates, $|\mathtt{C}| \geq \log n$, so $\mathtt{M} = 3 \cdot 2^{3|\mathtt{C}|} \geq 3n^3$. Note that Averaging also has an equivalence between converging and stabilizing: one all agents `ave` fields are within a certain interval, they cannot leave that interval. So by Lemma 3.8, Averaging, if it stabilizes, will stabilize to values of `ave` such that $\left\lfloor \frac{\mathtt{M}}{\mathtt{ave}} + \frac{1}{2} \right\rfloor = n$. We claim that Averaging stabilizes with probability 1, which is shown below. Furthermore, Timer reaches `MaxPhase` with probability 1, since the only way to avoid incrementing the phase of an agent is forever to avoid any interaction between it and an agent at the next phase, which happens with probability 0. This implies that with probability 1, the correct population size is eventually written into the field `count`.

It remains to show the claim that Averaging stabilizes with probability 1. Define the potential function Φ for any configuration $\vec{c}$ by $\Phi(\vec{c}) = \sum_a |a.\mathtt{ave} - \mathtt{M}/n|$, where the sum is over each agent a in the population. The Averaging protocol stabilizes by the time Φ reaches its minimum value,[16] which is either n or 0 depending on whether n divides $\mathtt{M}$, when all agents have ave $= \lfloor \mathtt{M}/n \rfloor$ or $\lceil \mathtt{M}/n \rceil$. We claim that Φ is nonincreasing with each transition of Averaging. When two agents meet, there are two cases: 1) both of their `ave` fields are $\geq \lceil \mathtt{M}/n \rceil$, or both are $\leq \lfloor \mathtt{M}/n \rfloor$, and 2) one of their `ave` fields is $<$ (resp., $\leq$) $\lfloor \mathtt{M}/n \rfloor$, and the other is $\geq$ (resp., $>$) $\lceil \mathtt{M}/n \rceil$. Taking the average of their `ave` fields, in case (1) does not change Φ, and in case (2) decreases Φ, so Φ is nonincreasing.

It remains to show that Φ will reach its minimum value with probability 1. If the protocol has not converged, then there must be some agent with an `ave` field not equal to either $\lfloor \mathtt{M}/n \rfloor$ or $\lceil \mathtt{M}/n \rceil$. But since the population-wide sum of the `ave` values is always $\mathtt{M}$, this implies that case (2) holds for some pair of agents. With probability 1, such a pair of agents must eventually meet, decreasing Φ. So with probability 1, Φ eventually reaches its minimum value. ◀

UniqueID stabilizes when all agents have a unique code since, by inspection of the UniqueID protocol, this implies that the codes no longer can change. The next lemma shows that this happens at most linear time.

▶ **Lemma 3.17.** UniqueID *stabilizes in expected time at most* $1.03n$.

Proof. By Lemma 3.2, once one agent reaches a new level, the expected time for all agents to reach that level is at most $4 \ln n$. Once all agents are at the same level and there is at least one pair of duplicate codes (i.e., UniqueID has not yet stabilized), the probability that the next interaction is a pair of agents with the same code is at least $1/\binom{n}{2} > 2/n^2$, so the expected number of interactions for these agents to meet and start a new level is at most $n^2/2$, so expected time $n/2$. Thus, once there is one agent at a level, the expected time to get an agent at the next level (assuming UniqueID does not stabilize at the current level) is at most $4 \ln n + n/2 < 0.51n$.

By Lemma 3.3, setting $\alpha = 1$, with probability at least $1 - \frac{5 \log \log n}{n}$, in $5 \ln n \log \log n$ time all agents reach a level k such that $\log n \leq k < 2 \log n$. Once there, if duplicate codes remain, it takes expected time $< 0.51n$ to reach level $2k$ by the above argument. If duplicate codes still remain, it similarly takes expected time $< 0.51n$ to reach level at least $4k$. By

[16] If $\mathtt{M} \gg 2n^2$, then Averaging can stabilize prior to this time, since the `ave` values do not have to reach their final convergent values for the output function $\left\lfloor \frac{\mathtt{M}}{\mathtt{ave}} + \frac{1}{2} \right\rfloor$ to converge, by Lemma 3.8.

Lemma 3.5, for any $j \geq 0$ (letting $\epsilon = j - 4$ in Lemma 3.5), duplicate codes remain at level $j \log n = (4 + \epsilon) \log n$ with probability at most $n^{-(j-4)}$.

Each new level after that takes $< 0.51n$ expected time. Thus the total expected time is at most (letting j above be 2^i below):

$$\begin{aligned}
&\underbrace{5 \ln n \log \log n}_{\text{time to reach level } k \geq \log n} + \underbrace{0.51n}_{\text{time to reach level } 2k \geq 2 \log n} + \underbrace{0.51n}_{\text{time to reach level } 4k \geq 4 \log n} \\
&\underbrace{+ \sum_{i=3}^{\infty} 0.51n \cdot \Pr[\text{level } 2^i \log n \text{ has duplicate codes}]}_{\text{contribution of levels} \geq 8 \log n} \\
\leq\ & 5 \ln n \log \log n + 1.02n + 0.51n \sum_{i=3}^{\infty} n^{-(2^i - 4)} && \text{Lemma 3.5} \\
<\ & 5 \ln n \log \log n + 1.02n + 0.51n \sum_{i=1}^{\infty} n^{-2^i + 2} \\
=\ & 5 \ln n \log \log n + 1.02n + 1 + 0.51n \sum_{i=2}^{\infty} n^{-2^i + 2} \\
<\ & 5 \ln n \log \log n + 1.02n + 1 + 0.51n \sum_{i=2}^{\infty} n^{-i} \\
<\ & 5 \ln n \log \log n + 1.02n + 1 + 0.51n \sum_{i=1}^{\infty} n^{-i} \\
=\ & 5 \ln n \log \log n + 1.02n + 1 + 0.51n \left(\frac{1}{1 - n^{-1}} - 1 \right) && \text{geometric series} \\
=\ & 5 \ln n \log \log n + 1.02n + 1 + 0.51n \frac{1}{n - 1} < 1.03n. \quad \blacktriangleleft
\end{aligned}$$

Proof of Theorem 3.15. By Theorem 3.14, ExactCounting converges to the correct answer in time $6 \ln n \log \log n$ with probability at least $1 - \frac{10 + 5 \log \log n}{n}$.

By Lemma 3.17, UniqueID converges in expected time at most $1.03n$. Once it has converged, it takes expected time $O(\log n)$ for Averaging to converge and Timer to write the correct output if it has not already been written. The sum of these times is at most $1.04n$ for sufficiently large n. We can bound the expected time as

$$\begin{aligned}
&\Pr[\text{convergence in time } \leq 6 \ln n \log \log n] \cdot 6 \ln n \log \log n + \\
&\Pr[\text{convergence in time } > 6 \ln n \log \log n] \cdot 1.04n \\
=\ & \left(1 - \frac{10 + 5 \log \log n}{n}\right) \cdot 6 \ln n \log \log n + \frac{10 + 5 \log \log n}{n} \cdot 1.04n \\
<\ & 6 \ln n \log \log n + 1.04(10 + 5 \log \log n) \\
<\ & 7 \ln n \log \log n. \quad \blacktriangleleft
\end{aligned}$$

3.7 Increasing time to minimize state complexity

ExactCounting generalizes straightforwardly to trade off time and memory: by adjusting the rate at which the code length grows, the convergence time $t(n) = O(f(n) \log n)$, where $f(n)$ is the number of stages required for the code length to reach $\log n$. The minimum state complexity is achieved when the code length increments by 1 each stage, so that $f(n) = \log n$ and $t(n) = \log^2 n$. In this case, a straightforward adaptation of Lemma 3.5, letting $\epsilon = 1$,

indicates that with probability at least $1 - \frac{1}{n}$, all codes are unique by level $3 \log n$. Carrying through the string length and integer bounds from the main argument gives state complexity $O(n^{30})$ for the full protocol and $O(n^9)$ for just the leader election.

3.8 Experiments

Simulations for the ExactCounting protocol are shown in Figure 2.

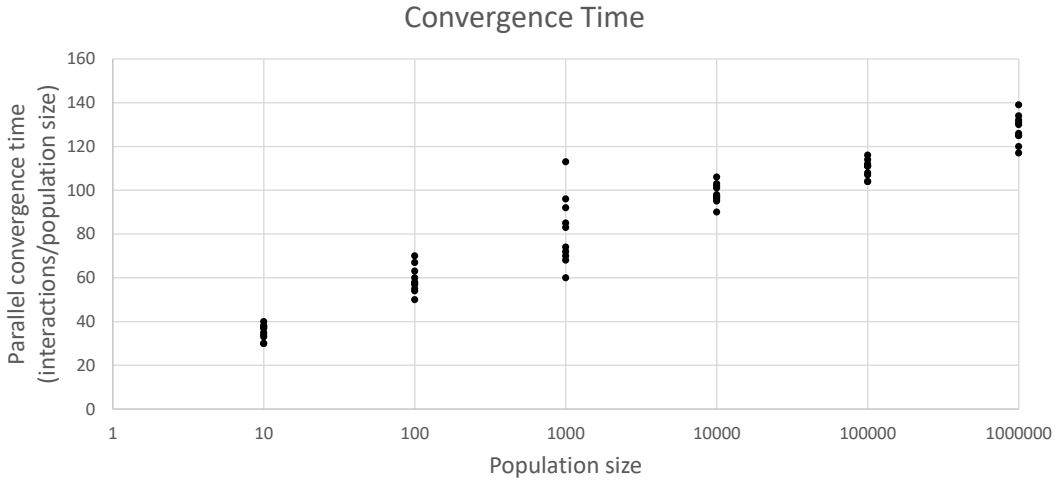


Figure 2 Simulated convergence time of ExactCounting. The dots indicate the convergence time of individual experiments. The population size axis is logarithmic, so exactly $c \log_{10} n$ time complexity would correspond to a line of slope c. Since $\log \log n$ is "effectively constant" (< 5) for the values of n shown, we similarly expect the plot to appear roughly linear.

4 Conclusion

We have shown a uniform population protocol computing the exact population size using $O(\log n)$ bits memory (i.e., poly(n) states) and $O(\log n \log \log n)$ time. By removing the Averaging and Timer subprotocols, the remainder is a uniform protocol electing a leader in $O(\log n \log \log n)$ time and $18 \log n$ bits of memory (for `C` and `LC`).

Some interesting questions are open. Is there a *uniform* polylogarithmic time population protocol, correct with high probability, for the problem of...

1. leader election, which is *terminating*?
2. constant-factor approximate size estimation, which is *terminating*?
3. exact size computation, which is *terminating*?
4. leader election, which is polylog(n) *state-bounded*?
5. constant-factor approximate size estimation, which is polylog(n) *state-bounded*?
6. exact size computation, which is $O(n)$ *state-bounded*?

For Question 6, the trivial lower bound is n, but the Averaging protocol seems intuitively to require $\Omega(n^2)$ states. It would be interesting to prove a $\Omega(n^2)$ lower bound, either for "any scheme based on averaging" (suitably formalized), or more generally for any sublinear-time size counting protocol. Since Averaging requires $O(n^2)$ states, if the initial

configuration has a leader and a constant-factor approximation of n, this means that solutions to questions 1, 2, 4, and 5 would immediately imply a $O(n^2)$ state, polylog(n) time protocol for exact population size computation.

Since many problems such as leader election require only an estimate on the population size, not an exact value, a protocol answering questions 2 and 5 is an important goal. Alistarh, Aspnes, Eisenstat, Gelashvili, Rivest [1] have shown a uniform protocol (converging, but not terminating) using only polylog(n) states that in $O(\log n)$ expected time can get an estimate n' within a polynomial (but not constant) factor of the true size n: with high probability $1/2 \log n \leq \log n' \leq 9 \log n$ i.e., $\sqrt{n} \leq n' \leq n^9$.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *SODA 2017: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2560–2579. SIAM, 2017.

2 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *SODA 2018: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2221–2239. SIAM, 2018.

3 Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9135 of *Lecture Notes in Computer Science*, pages 479 – 491. Springer, Berlin, Heidelberg, 2015.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, March 2006.

5 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *25th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, New York, NY, USA, 2006. ACM Press.

6 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, September 2008.

7 James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. Time and Space Optimal Counting in Population Protocols. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70, pages 13:1–13:17, 2017.

8 Joffroy Beauquier, Janna Burman, Simon Claviere, and Devan Sohier. Space-optimal counting in population protocols. In *DISC 2015: International Symposium on Distributed Computing*, pages 631–646. Springer, 2015.

9 Joffroy Beauquier, Julien Clement, Stephane Messika, Laurent Rosaz, and Brigitte Rozoy. Self-stabilizing counting in mobile sensor networks with a base station. In *Distributed Computing*, pages 63–76. Springer Berlin Heidelberg, 2007.

10 Amanda Belleville, David Doty, and David Soloveichik. Hardness of computing and approximating predicates and functions with leaderless population protocols. In *ICALP 2017: 44th International Colloquium on Automata, Languages, and Programming*, volume 80 of *LIPIcs*, pages 141:1–141:14, 2017.

11 Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and Efficient Leader Election. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61, pages 9:1–9:11, 2018.

12 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *PODC 2017: Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 451–453. ACM, 2017.

13 James M Bower and Hamid Bolouri. *Computational modeling of genetic and biochemical networks*. MIT press, 2004.

14 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

15 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.

16 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*. to appear. Special issue of DISC 2015 invited papers.

17 Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *SODA 2018: ACM-SIAM Symposium on Discrete Algorithms*, 2018. to appear.

18 Shafi Goldwasser, Rafail Ostrovsky, Alessandra Scafuro, and Adam Sealfon. Population stability: regulating size in the presence of an adversary. In *PODC 2018: Proceedings of the ACM Symposium on Principles of Distributed Computing*, 2018. to appear.

19 Tomoko Izumi, Keigo Kinpara, Taisuke Izumi, and Koichi Wada. Space-efficient self-stabilizing counting population protocols on mobile sensor networks. *Theor. Comput. Sci.*, 552:99–108, 2014.

20 Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 102–109, 2004.

21 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM symposium on Theory of computing (STOC)*, pages 513–522. ACM, 2010.

22 Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. Counting in anonymous dynamic networks under worst-case adversary. *IEEE 34th International Conference on Distributed Computing Systems (ICDCS)*, 2014.

23 Othon Michail. Terminating distributed construction of shapes and patterns in a fair solution of automata. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 37–46, 2015. Also in *Distributed Computing*, 2017.

24 Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Naming and counting in anonymous unknown dynamic networks. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 281–295. Springer, 2013.

25 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

26 Y. Mocquard, E. Anceaume, and B. Sericola. Optimal proportion computation with population protocols. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 216–223, Oct 2016.

27 Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In *14th IEEE International Symposium on Network Computing and Applications*, pages 35–42, 2015.

28 Lulu Qian, David Soloveichik, and Erik Winfree. Efficient Turing-universal computation with DNA polymers. In *DNA 2010: Proceedings of The Sixteenth International Meeting on DNA Computing and Molecular Programming*, volume 6518 of *Lecture Notes in Computer Science*. Springer, 2010.

29 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

30 David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107:5393–5398, 2010.

31 Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358(6369):eaal2052, 2017.

32 Chris Thachuk and Anne Condon. Space and energy efficient computation with DNA strand displacement systems. In *DNA 2012: 18th International Conference on DNA Computing and Molecular Programming*, pages 135–149. Springer, 2012.

33 Vito Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Mem. Acad. Lincei Roma*, 2:31–113, 1926.