Theoretical Computer Science 894 (2021) 91–102

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A survey of size counting in population protocols

David Doty [1], Mahsa Eftekhari [*,1]

University of California, Davis, United States of America

ARTICLE INFO

Article history:
Received 26 March 2021
Received in revised form 25 August 2021
Accepted 29 August 2021
Available online 1 September 2021

Keywords:
Population protocols
Population size counting
Exact counting
Approximate counting

ABSTRACT

The population protocol model describes a network of n anonymous agents who cannot control with whom they interact. The agents collectively solve a computational problem through random pairwise interactions, each agent updating its state in response to seeing the state of the other agent. Population protocols are equivalent to the model of chemical reaction networks, describing abstract chemical reactions such as $A + B \to C + D$, when the latter is subject to the restriction that all reactions have two reactants and two products, and all rate constants are 1. The *counting* problem is that of designing a protocol so that n agents, all starting in the same state, eventually converge to states where each agent encodes in its state an exact or approximate description of population size n. In this survey paper, we describe recent algorithmic advances on the counting problem.

1. Introduction

A population is a network of n anonymous and identical *agents*, each holding a *state* representing its entire memory. The agents communicate through a sequence of randomly chosen pairwise interactions. In an interaction, the scheduler selects two different agents uniformly at random. Each observes the state of the other and updates its own according to the transition function defined by a protocol. A protocol is designed to perform a common task, e.g., leader election: selecting exactly one agent as leader over the population. Following the protocol drives the population from a valid initial configuration to a desired configuration (for example, from all agents being leaders to only one).[2]

A protocol is defined by *transitions* that describe, given a pair of input states of the agents that interact, how the agents should update their memory. For example, a simple leader election transition is $(L, L) \to (L, F)$ with all agents starting in state L. This protocol *stabilizes*, meaning that with probability 1, it reaches a configuration that is both correct and *stable*: every subsequently reachable configuration is also correct. In the original model of population protocols [4], the states and transitions are constant with respect to the population size n. However, recent studies use a variant of the model: allowing the number of states and transitions to grow with n. One motivation to study population protocols with $\omega(1)$ states is the existence of impossibility results showing that no constant-state protocol can stabilize in sublinear time with probability 1 for problems such as leader election [5], majority [6], or computation of more general predicates and integer-valued functions [7].[3] The recent algorithmic advances using non-constant states [6,8–20], lead to time- and space-optimal

* Corresponding author.
E-mail addresses: doty@ucdavis.edu (D. Doty), mhseftekhari@ucdavis.edu (M. Eftekhari).

[1] Supported by NSF award 1900931 and CAREER award 1844976.

[2] For the problems of leader election and population size computation, all agents start in the same state in a valid initial configuration. However, in some settings, the initial configuration is assumed already to contain a unique leader [1–3].

[3] It is conventional in population protocols to measure the memory usage by counting the total number $s(n)$ of states agents can store in population size n, instead of the number of bits required to represent these states, which is about $\log s(n)$.

https://doi.org/10.1016/j.tcs.2021.08.038
0304-3975/

solutions for leader election [19] and majority [18] problems. However, most of these solutions [6,8,9,11–20] propose a nonuniform protocol. Rather than being a single set of transitions, a nonuniform protocol represents a *family* of protocols, where the set of transition rules (i.e., the protocol) used is allowed to depend on the population size n.

The typical way that nonuniformity appears in a protocol is that the agents have an estimate of n (for example the value $\lceil \log n \rceil$) appearing in the transitions. When expressed in pseudocode, this is often realized by a "hard-coded" constant $\lceil \log n \rceil$ to which the code has access (i.e., each agent receives the value $\lceil \log n \rceil$ as "advice"). However, in measuring the state complexity of the protocol, the space required to store this value does not count against the memory usage. Note that this concern is relevant because we count memory complexity by counting the number of states, rather than the number of bits necessary to represent the state. Adding a field with $O(\log n)$ different values does not asymptotically change the bit usage, but it does asymptotically increase the number of states. As an example of a nonuniform protocol using its estimate of $\log n$, consider the following nonuniform leaderless phase clock rules: each agent, independently, counts its number of interactions and uses it as a timer by comparing to $12 \ln n$. In this protocol, each agent increment the value in its `count` field until it reaches a threshold dependent on $\ln n$ (note that "$-$" stands for an arbitrary value in $\mathbb{N}$ for `count`):

$$\begin{cases} (\mathtt{A} = \mathsf{F}, \mathtt{count} = i), (-,-) \longrightarrow (\mathtt{A} = \mathsf{F}, \mathtt{count} = i+1), (-,-) & \textbf{if } i < 12 \ln n \\ (\mathtt{A} = \mathsf{F}, \mathtt{count} = i), (-,-) \longrightarrow (\mathtt{A} = \mathsf{T}, \mathtt{count} = 0), (-,-) & \textbf{if } i = 12 \ln n \end{cases} \tag{1}$$

In the above leaderless phase clock, no agent will set their `A` field to `T` before $6 \ln n$ time has passed with probability at least $1 - 1/n$.[4] However, no uniform protocol can achieve this same task: in any uniform protocol, some agent will set its `A` field to `T` in constant time with high probability [21, Theorem 4.1].

Uniform computation In a *uniform* protocol, by contrast, the transitions are not dependent on the population size n, i.e., agents lack any knowledge of n.

The original $O(1)$-state model [1,4,22] is uniform, since there is a single transition function for all population sizes. However, uniform protocols are not required to have constant states. For example, starting from n agents in state L_1, the protocol defined by transitions $L_i, L_j \to L_{i+j}, F$ for all natural numbers i, j, in a population of size n, can produce all values of L_i for i between 1 and n. However, note that the (infinitely many) transitions are "uniformly specified": no transition makes reference to an estimate of n. (This is formalized by requiring the transition function to be computable by a single Turing machine [23,24].)

1.1. Definitions and notation

To measure a protocol's computation time, we consider the expected number of interactions, starting from the initial configuration to reach a desired configuration. Since we would like to model that many interactions can happen in parallel, with $O(1)$ interactions per agent per unit of time, we define n interactions as one unit of time. This definition coincides with the time defined in the standard Gillespie kinetic model for chemical reaction networks [25], of which population protocols are a special case describing n molecules reacting in a volume proportional to n.

We say that a protocol *stably* solves a problem if the agents eventually reach a correct configuration with probability 1, and no subsequent interactions can move the agents to an incorrect configuration; i.e., the configuration is *stable*.

In this paper, we use the term "with high probability" (or w.h.p.) to refer the probability of at least $1 - 1/n$. However, the standard definition of high probability refers to probability of at least $1 - 1/n^c$ for some constant $c > 0$, where c can be made arbitrarily large by adjusting appropriate parameters in the algorithm.

Since in each interaction, the scheduler picks an *ordered* pair of agents to interact, we denote these agents in the pseudocode as receiver (`rec`) and sender (`sen`) respectively. In other words, unlike many models of distributed computing, population protocols typically are defined to be able to break symmetry "for free".[5]

1.2. Population size counting

Population size counting is the problem of computing the number of agents in a population protocol. Both exact [24, 27] (computing n) and approximate counting [2,6,21,27] (computing $\lceil \log n \rceil$ or $\lfloor \log n \rfloor$, which gives $2^{\lceil \log n \rceil}$ or $2^{\lfloor \log n \rfloor}$ as a multiplicative factor-2 estimate of n) have been considered in the literature. Considering the size counting problem in a nonuniform model of population protocol is trivial since we can provide the agents the values of n or $\lceil \log n \rceil$ as advice. Thus, all cited papers solving this problem use the uniform variant of the model [2,6,21,24,27].

Motivation The recent algorithmic advances for population size counting problem provide composable building blocks that simplify the (uniform) solution of other problems: compute an estimate of $\log n$, and use this value where a nonuniform

[4] We can compute the error probability with a straightforward Chernoff bound application on binomial random variables.

[5] There are examples of interesting protocols using only symmetric transitions, and under certain circumstances, asymmetric protocols can be simulated by symmetric ones [26].

Table 1
Summary of existing protocols for the exact counting problem. Note that "stable" means correct with probability 1. For all the stable protocols, the stated time bounds the stated time bounds are proven both with high probability and in expectation. However, the state complexity for all the protocols is correct with high probability. We also mention the correctness probability for each protocol under the "prob." column. We also discuss counting in population protocols with constant message size and in the self-stabilizing model in Sections 3.4, 3.5 respectively.

Exact counting protocols					
Ref.	Sec.	Prob.	Time	States	Comments
[24]	3.2	1	$O(\log n \log\log n)$	$O(n^{60})$	stable
[27]	3.3	$1-\frac{O(1)}{n}$	$O(\log n)$	$O(n \log n)$	-
[27]	3.3	1	$O(\log n)$	$O(n \log n \log\log n)$	stable

Table 2
Summary of existing leaderless protocols for the approximate counting problem. The approximation factor of each protocol is implied under the "output value". The columns follow the same convention as Table 1. One leader-driven protocol is discussed in Section 4.5.

Approximate counting protocols						
Ref.	Sec.	Output value (range)	Prob.	Time	States	Comments
[18,27]	4.1	$\lfloor \log n \rfloor$	1	$O(n \log n)$	$O(\log^2 n)$	stable
[6]	4.2	$[\frac{1}{2}\log n, 9\log n]$	$1-\frac{O(1)}{n^3}$	$O(\log n)$	$O(\log^9 n)$	deterministic
[6,15,21]	4.2	$[\log n - \log\ln n, 2\log n]$	$1-\frac{O(1)}{n}$	$O(\log n)$	$O(\log^2 n)$	-
[10,20]	4.3	$[\frac{\log n}{16}, 256\log n]$	$1-\frac{O(1)}{n}$	$O(\log n)$	$O(\log\log n)$	deterministic
[21]	4.4	$[\log n - 5.7, \log n + 5.7]$	$1-\frac{O(1)}{n}$	$O(\log^2 n)$	$O(\log^4 n)$	-
[27]	4.6	$\lfloor \log n \rfloor$ or $\lceil \log n \rceil$	$1-\frac{O(\log n)}{n^2}$	$O(\log^2 n)$	$O(\log n \log\log n)$	-
[27]	4.6	$\lfloor \log n \rfloor$ or $\lceil \log n \rceil$	1	$O(\log^2 n)$	$O(\log^2 n \log\log n)$	stable

protocol would use the hard-coded constant $\lceil \log n \rceil$. We can adopt a counting technique as a black box and compose it with a nonuniform protocol through a restarting scheme [10,21,27] to obtain a uniform protocol. We explain the composition scheme in Section 5.

In this survey paper, we will discuss the existing counting protocols and draw attention to their time-space tradeoff. We will cover both the exact and approximate counting problems, since, for most protocols, having an approximation of $\log n$ suffices. Tables 1, 2 summarize both exact and approximate counting protocols in the conventional model of population protocols (with initialized population).

2. Prerequisite: fast averaging protocol

The averaging technique discussed in this section does not solve the counting problem but is used in the subsequently discussed counting protocols in Sections 3.2, 3.3, and 4.6.

The averaging technique, also known as randomized load balancing [28–30], was first introduced in population protocols in [31] to solve the exact majority problem.

Each agent's state is an integer; for intuition, assume the integers represent a "load"[6] that each agent holds. The averaging rules allow each selected pair of agents to exchange loads to balance (as best they can) their values, e.g., $(2, 11) \to (6, 7)$. This leads the population to a configuration in which all agents have almost equal values: concretely, if the total load among the population is m, then after stabilization, each agent holds either the value $\lfloor m/n \rfloor$ or $\lceil m/n \rceil$. Stabilization can take $\Theta(n)$ time in the worst case, but it takes only $O(\log n)$ time for all agents to hold *three* consecutive values (two of which are $\lfloor m/n \rfloor$ or $\lceil m/n \rceil$) [28,32]. The averaging technique has been crucial in several polylogarithmic-time protocols for problems such as population size counting [24,27] and majority related problems [13,17,31,32], and its time complexity has been tightly analyzed [17,28,29,33].

Notably, Mocquard, Anceaume, Aspnes, Busnel, and Sericola [17] used the averaging technique to solve a generalization of the exact majority problem. Considering a population with n_a, n_b initialized agents in states `A` and `B`, i.e., $n_a + n_b = n$, the authors designed an averaging-based protocol that counts the *exact* difference between the number of agents in the `A` and `B` (computing the value of $n_a - n_b$).

In this protocol, the `A` and `B` agents start with $+m$ and $-m$ values respectively, where m is a large integer with respect to the population size n. Thus the population as a whole starts with a total of $m(n_a) - m(n_b)$ load. The protocol is designed to almost equally distribute the load among the agents while *preserving the total sum*. In this protocol, the agents update their state according to the "discrete averaging" rule described in Protocol 1.

[6] In the rest of the paper, whenever the load values are nonnegative, we use "tokens" instead to present a more intuitive explanation of the protocols.

Protocol 1 `DiscreteAveraging(rec, sen)`.

Initialization:
if `agent.input` = `A`: `agent.average` $\leftarrow m$
if `agent.input` = `B`: `agent.average` $\leftarrow -m$
1: `rec.average, sen.average` $\leftarrow \left\lceil \frac{\texttt{rec.average}+\texttt{sen.average}}{2} \right\rceil, \left\lfloor \frac{\texttt{rec.average}+\texttt{sen.average}}{2} \right\rfloor$
2: `rec.output` $\leftarrow \left\lfloor \frac{n \times \texttt{rec.average}}{m} + \frac{1}{2} \right\rfloor$

Protocol 2 `IntegerTokenPassing(rec, sen)`.

Initialization:
`agent.count` $\leftarrow 1$; `agent.active` $\leftarrow$ True
1: **if** `rec.active` = True & `sen.active` = True **then**
2: `rec.count, sen.count` $\leftarrow$ `rec.count` + `sen.count`
3: `sen.active` $\leftarrow$ False
4: **if** `rec.active` = False & `sen.active` = False **then**
5: `rec.count, sen.count` $\leftarrow$ max(`rec.count, sen.count`)

Initializing Protocol 1 with n_a, n_b agents in states `A` and `B`, it is shown that the agents' `average` value converges to $\frac{(n_a-n_b)m}{n}$ quickly. In fact, the authors of [17] proved for $m = \left\lceil \frac{\sqrt{2}n^{3/2}}{\sqrt{\delta}} \right\rceil$, after $O(n \log n)$ interactions with probability $1-\delta$, the `output` field of agents will be equal to $n_a - n_b$ (e.g. to achieve a high probability result, one can set $\delta = 1/n$ and start the protocol with $m = O(n^2)$).

The protocol given in [17] is nonuniform; it is assumed that the population size is known in advance. Crucially, the protocol requires all agents to store the exact value of n in their memory to compute the `output`. In a separate paper [32], Mocquard, Anceaume, and Sericola show how to remove this assumption and make the protocol uniform. Their protocol computes the ratio of `A` agents with respect to n within a multiplicative factor error $(1+\epsilon)$ of the true proportion for any precision $\epsilon > 0$ using $2\left\lceil \frac{3}{4\epsilon} \right\rceil + 1$ states.

3. Exact population size counting

In the exact size counting problem, the agents aim to compute their population size n. In some protocols, to reduce the space complexity, the agents report their estimate of n as a function of their internal fields [24,27] rather than storing the population size explicitly in their memory. This trick helps the agent to describe numbers that exceeds their memory limit. For example, the agents might store $\texttt{a} = \lfloor \log n \rfloor$ but set their output as $2^{\texttt{a}}$ without explicitly computing the value of $2^{\texttt{a}}$ to keep their memory usage $\Theta(\log n)$ instead of using linear states.

3.1. Naïve slow protocol

A naïve protocol can count the number of agents in a population using a modified version of the slow leader election protocol. All of the agents start in the `active` state holding 1 token in their `count` variable. For consistency with other counting protocols in this section, we name the leader `active`, retaining the standard pairwise leader elimination (`active`, `active`) $\rightarrow$ (`active`, not `active`). We also change the leader election protocol so that the final remaining `active` agent accumulates all the n tokens. The rules of this protocol preserve the total sum of the active tokens. When two active agents interact one of them becomes inactive, and both change their `count` value to the sum of their accumulated tokens. Initially, the agents start with n scattered tokens and eventually, there will remain one `active` agent having all the tokens. Protocol 2 describes how the agents update their state at each interaction.

The transitions of Protocol 2 require $O(n)$ time to converge to the exact population size n.[7] $\Omega(n)$ is a clear lower bound on the number of states needed for any protocol that requires agents to store the value n, since $\lceil \log n \rceil$ bits are required merely to write the number n. Protocol 2 solves this problem using $2n$ states.

3.2. Fast exact counting with polynomial states

Doty, Eftekhari, Michail, Spirakis, and Theofilatos [24] devised the first sublinear time protocol for the exact population size counting problem. Although their protocol heavily relies on the idea of the averaging protocol of [17] (explained in Section 2), they managed to eliminate the "advance knowledge of n" assumption. Their protocol achieves uniformity by putting together one phase of leader election and approximate counting before the averaging phase. Additionally, the averaging part of [24] is slightly different from the protocol of [17] by fixing the number of agents in groups `A` and `B` and changing their initial values. Recall that in Protocol 1, the `A`, `B` agents start with $+m$, $-m$ respectively and they converge to

[7] This is a standard analysis in population protocols; for instance, see [4, Section 6]. One way to see it requires $\Omega(n)$ time is to observe that once exactly two agents have `active` = True, since there are $\binom{n}{2} = \Theta(n^2)$ total pairs of agents, it takes expected $\Theta(n^2)$ interactions, i.e., $\Theta(n)$ expected parallel time, for the two `active` agents to interact and reduce their count to one.

$\frac{(n_a-n_b)m}{n}$. In the protocol of [24], the agents start the averaging phase in a very special case of one A agent with $+m$ tokens and $n-1$ of B agents with 0 tokens. Following the rules of Protocol 1, the agents' average value will converge to $\approx m/n$. The authors proved with any values of $m \geq 3n^3$, $\lfloor \frac{\texttt{average}}{m} + \frac{1}{2} \rfloor$ will be equal to n after $O(\log n)$ time. Berenbrink, Kaaser, and Radzik improved this result showing that the correctness of the above statement holds for smaller values of m as long as $m \geq 4n^2$ [27, Lemma 4.2], which implies a better space complexity since we can initialize the agents with smaller values of m in the averaging protocol, preserving the correctness of the output.

It remains to show how to initialize the population with the above requirement: having one agent in group A, called leader, with $m \geq 3n^3$ tokens. To achieve this, the protocol of [24] begins by assigning unique codenames (binary strings), of the same length, to all the agents. All agents start with the empty string ε as their codename. New codenames are generated dynamically whenever two agents with the same codename $= x_1 x_2 \dots x_l$ of length l interact; each decides a new codename of length $2l$ by appending l more random bits to their codename. Also, if one agent has a longer codename than its partner, the latter appends random bits until their codename lengths are the same. Once the agents have unique codenames of length l^*, it is shown that $\log n \leq l^* \leq 3 \log n$ holds with high probability. This provides each agent with a polynomial-factor approximation of n that is in $[n, n^3]$. The leader election protocol of [24] is as simple as a pairwise comparison of codenames, adopting the lexicographically largest codename as the leader. Once there is a unique leader and an estimate of n, the agents start the averaging subprotocol. This protocol as designed is stabilizing (correct with probability 1) and converges to the correct value of n after $\Theta(\log n \log\log n)$ time both w.h.p. (probability at least $1 - \frac{O(\log\log n)}{n}$) and in expectation. The error of having multiple leaders always will be detected (since all agents eventually have unique codenames) and the agents replace their output value resulting an always correct protocol.

3.3. Fast exact counting with linear states

Berenbrink, Kaaser, and Radzik [27] improved the space complexity as well as the time complexity of the exact counting protocol of [24]. They start their exact counting with a subprotocol of [10] that elects not a single leader but a "junta" of n^ϵ leaders, for $0 \leq \epsilon < 1$. In the junta election protocol of [10] (see Section 4.3 for more details), each agent computes a level value, and the maximum level among the agents is an approximation of $\log\log n$: assuming l^* is the maximum level, $\log\log n - 4 \leq l^* \leq \log\log n + 8$ with probability at least $1 - O(1/n)$ [20, Lemma 4] [10, Theorem 3]. Having a junta of size n^ϵ, opens the possibility of simulating a "phase clock" that allows agents to stay synchronized within phases of length $\Theta(\log n)$ for poly(n) time [1,10,20,34]. In addition to the junta, the exact counting protocol of [27] requires having one unique leader.

They use the leader election protocol from [20] that uses constant number of phases to elect a leader: in every even phase, each remaining leader generates a sequence of $\Theta(\log n)$ random bits. In the odd phases, they broadcast the maximum bitstring by epidemic and if a leader encounters a larger bitstring than its own, it updates its state to follower. This leader election protocol is a generalization of the $O(\log^2 n)$ time protocol described in [10], which allows remaining leaders to generate and broadcast 1 random bit in each phase and continues for $O(\log n)$ phases of each $O(\log n)$ time.

In the rest of the protocol, we assume there exists a leader and the agents all hold the value l^* computed as described above. Moreover, the agents are synchronized via the junta-driven phase clock that gives them phases of $\Theta(\log n)$ time. Note that, the averaging process explained in Protocol 1 takes $\Theta(\log n)$ time to almost equally distribute the initialized load. At this point, it is possible to adopt the technique of [24] explained in Section 3.2 and, using the fact that $2^{2^{l^*+4}} \geq n$, initialize the leader with at least n^2 tokens. However, this approach leads to a protocol that uses at least $2^{2^{\log\log n+12}} = 2^{4096 \log n} = n^{4096}$ states (when $l^* = \log\log n + 8$), which is worse than $O(n^{60})$ states of the protocol of [24]. In [27], the authors refine the approximation value of $\log n$ through possibly multiple (constant) phases of $O(\log n)$ time each, that eventually a total of at least $2n$ tokens will be distributed among agents:

The leader initializes the averaging process with $2^{2^{l^*-8}}$ tokens (note that $2^{2^{l^*-8}} \leq n$) and signals the agents to multiply the total number of tokens by $2^{2^{l^*-8}}$ followed by an averaging phase until the total number of distributed tokens is less than $2n$.

Specifically, by the end of each averaging phase, if the leader's average is less than 4, all the agents (including leader) multiply their average by another $2^{2^{l^*-8}}$ and repeat the averaging process. Note that this will multiply the total number of tokens by $2^{2^{l^*-8}}$. Depending on the precision of l^* for $\log\log n$, this process may take multiple (constant) phases of multiplication followed by an averaging phase, and stops once the leader has average ≥ 4. At this point, the leader computes an approximation of $\log n$ (stores in k) as a function of $(p_i, l^*, \texttt{average})$ (precisely, set $k = p_i \cdot 2^{l^*-8} - \lfloor \log(\texttt{average}) \rfloor$) in which p_i indicates how many times the agents multiplied their average value. The authors proved $\log n - 3 \leq k \leq \log n + 3$ holds w.h.p.

In the next stage of the protocol, the agents compute the exact value of n using the computed k value as an approximation of $\log n$ via two phases of averaging:

The leader broadcasts k, to all agents and initializes a new averaging process with $c \cdot 2^k$ tokens where $c = 2^8$ and is a constant. The agents distribute $c \cdot 2^k$ tokens through the averaging phase and by the end of it, all agents multiply their average value by 2^k (once) and repeat the averaging. By the end of this phase, a total of at least n^2 tokens has been distributed. Thus, the agents can compute the exact value of n, similar to [24], as a function of $(c \cdot 2^{2k}, \texttt{average})$.

In contrast to the protocol of [24], where the leader starts with poly(n) tokens (n^c for $3 \leq c \leq 9$), at every stage of the protocol of [27], the leader starts with no more than n tokens. Once the agents have almost equal tokens because of the averaging phase, the entire population multiplies their `average` value (tokens) by another factor of $\approx n$. This trick puts an upper bound of n over the range of possible values of `average` but achieves having a total of poly(n) tokens among the population. The protocol of [27] uses $O(n \log n)$ states and converges in $O(\log n)$ time both w.h.p. However, this protocol has a small probability of error; i.e., it is not *stable* (See Section 1.1). It is explained in [27] how to achieve stabilization in $O(\log n)$ time using $O(n \log n \log\log n)$ states with error detection schemes that point agents to switch to the naïve slow (but stable) Protocol 3.1 as a backup.

3.4. Population protocols with constant size messages

Amir, Aspnes, Doty, Eftekhari, and Severson [35] studied the exact counting problem in population protocols with large memories but limited (constant) message size. Considering the exact population size counting problem in this model, the authors of [35] proposed a leader-driven protocol to count the exact population size that converges in $O(\log^2 n)$ time using $O(n \log^2 n)$ states with probability at least $1 - O(1/n)$. They also proposed a leaderless protocol that counts the exact number of agents in a population using $O(\log^2 n)$ time and $O(n \operatorname{polylog} n)$ states with probability at least $1 - O(1/n)$. They also demonstrated protocols that *approximate* the population size, also using $O(1)$ messages. See Section 4 for a definition of approximate population size counting.

The following large-message protocol allows agents to compute n: the leader starts with value 1, and agents conduct a rational-number variant of the averaging protocol (e.g., $1, 0 \to \frac{1}{2}, \frac{1}{2}$; $\frac{1}{2}, 0 \to \frac{1}{4}, \frac{1}{4}$; $\frac{1}{4}, \frac{1}{8} \to \frac{3}{16}, \frac{3}{16}$) until all agents hold dyadic values close enough to $\frac{1}{n}$ that they can uniquely identify the size n. The protocol of [35] simulates this in $O(\log n)$ phases (synchronized via a leader-driven phase clock), averaging together only *constantly* many values at a time, narrowing the interval of values stored internally by agents, until it contains a unique integer reciprocal $\frac{1}{n}$.

3.5. Self-stabilizing counting

So far we have discussed the *initialized* setting, where we assume the protocol is permitted to designate a set of *valid* initial configurations. In the case of the counting problem, we identify a special state x_0, where valid initial configurations have all agents in state x_0. In contrast, in the *self-stabilizing* setting, once the set of states has been defined by the protocol, an adversary can initialize the population with an *arbitrary* configuration assigning these states to agents. This is an extreme form of fault tolerance, modeling errors that can alter states arbitrarily, at any time during the execution of a protocol, requiring the protocol to be able to recover from any number of such transient errors, by considering the "initial" configuration to be the (arbitrary) configuration just after the *last* such transient error.

It is worth observing why counting, as defined previously, is impossible in this strict setting. Suppose that a population of n agents has stabilized on output n. Then for any $k < n$, in the self-stabilizing setting, any configuration of a sub-population of k of these agents is a valid starting configuration for population size k. Then this size-k population must eventually change their output from n to k. However, the interactions that achieve this are possible in the size-k sub-population of the original size-n population, contradicting its stability.

To circumvent this impossibility, protocols for the self-stabilizing counting problem have considered adding one exceptional entity, called the *base station*, such that the adversary is not permitted to affect its memory [36–39]. Furthermore, only the base station is required to know the count after stabilization; thus it is possible for other agents to have fewer than n states. In these protocols, the base station stably computes the exact number n of agents in the population, called *counted* agents. Assuming a known upper bound P on the population size n, Beauquier, Clement, Messika, Rosaz, and Rozoy [38] proposed a protocol that solves the exact counting problem using $4P$ states. This result improved by Izumi, Kinpara, Izumi, and Wada [39] to $2P$ states space per counted agent. In both protocols, the base station assigns unique names to the counted agents. Beauquier, Burman, Clavière, and Sohier [37] proposed a space-optimal protocol that solves the exact counting problem using 1-bit memory for each counted agent in $O(2^n)$ time. Later on, Aspnes, Beauquier, Burman, and Sohier reduced the exponential time complexity to $O(n \log n)$ time which is also proven to be optimal while still using 1-bit memory for the counted agents [36].

4. Approximate population size counting

The study of the counting problem is partially motivated by the existence of nonuniform protocols. Most of these nonuniform protocols require not n exactly, but an approximation, e.g., the value $\lceil \log n \rceil$. In the approximate size counting problem, the agents compute an approximation of n, e.g., $2^{\lceil \log n \rceil}$, the smallest power of two greater than n, rather than the exact value n. This freedom opens room for protocols with exponentially smaller space complexity.

4.1. Naïve slow protocol

Recall that Protocol 2 solves the exact counting problem via pairwise elimination of active agents, passing all the tokens (where each agent starts with one token) to the remaining active agent. A simple modification to Protocol 2 can solve the

Protocol 3 `PowersOfTwoTokenPassing(rec, sen)`.

Initialization:
`agent.exponent` $\leftarrow 0$; `agent.active` $\leftarrow$ True

```
1: if (rec.active & sen.active) & (rec.exponent = sen.exponent) then
2:     rec.exponent, sen.exponent ← rec.exponent + 1
3:     sen.active ← False
4: if rec.active = False & sen.active = False then
5:     rec.exponent, sen.exponent ← max(rec.exponent, sen.exponent)
```

approximate counting problem using $O(n \log n)$ time [18,20]. In the protocol presented next, token counts are restricted to powers of two, thus using only $\Theta(\log n)$ states. All agents start in the `active` state with one token stored in their `exponent` field (initially set to 0 representing integer 2^0). When two `active` agents with the same `exponent` value equal to i (integer value of 2^i) interact, one of them becomes not `active`, and both update their `exponent` to $i+1$ (integer value of 2^{i+1}). Additionally, all not `active` agents help propagating the maximum value of `exponent` they have seen (described in Protocol 3).

Although Protocol 3 is slow and takes $O(n \log n)$ time, it utilizes almost optimal space complexity. This protocol uses $O(\log n)$ states having $n - O(\log n)$ agents store $\lfloor \log n \rfloor$; however, requiring all agents to store $\lfloor \log n \rfloor$ results in a $O(\log^2 n)$ state protocol.[8] Note that $\lceil \log \log n \rceil$ bits (equivalently $\Theta(\log n)$ states) are needed to write the number $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$ for any protocol that reports an estimation of $\log n$ as its output.

In the following, we overview the fast protocols that considered the approximate counting problem. Commonly, the output of these protocols is an approximation of $\log n$.

4.2. Maximum of n geometric random variables

Assuming a randomized protocol, i.e., agents have access to independent, unbiased random bits, there is a simple method for obtaining a constant-factor approximation of $\log n$, i.e., a polynomial factor approximation of n. Recall that a $\frac{1}{2}$-*geometric* random variable is the number of flips of a fair coin until the first heads. It is known that the maximum of n independent $\frac{1}{2}$-geometric random variables is in the interval $[\log n - \log \ln n, 2 \log n]$ with probability at least $1 - O(1/n)$ [21,40]. Each agent flips a fair coin on each interaction, incrementing a counter until the first heads,[9] and then moves to a "propagate the maximum" stage where the maximum counter value obtained by any agent is spread by epidemic throughout the population, i.e., $i, j \to i, i$ if $i > j$.

4.2.1. Synthetic coins

Since it may be desirable to use a deterministic transition function, some work has been done on techniques for simulating randomized transitions with a deterministic transition function. Alistarh, Aspnes, Eisenstat, Gelashvili, Rivest [6] proposed a general technique, known as *synthetic coins*, that synthesizes "almost" independent and unbiased random coin flips in a deterministic protocol, "extracting" randomness from the random scheduler. Each agent uses this synthetic coin technique to simulate generating a $\frac{1}{2}$-geometric random variable G_i. Their protocol provides an approximation of $\log n$ in the interval $[1/2 \log n, 9 \log n]$ with probability at least $1 - O(1)/n^3$, i.e., worse bounds than obtained with independent, unbiased coin flips, but still within a constant factor of $\log n$.

Recently, Sudo, Ooshita, Izumi, Kakugawa, and Masuzawa [15] proposed an improved implementation of synthetic coins: independent and unbiased coins (as with [6], using only symmetric transitions). The method of Sudo et al. [15] works as follows for any protocol where "population splitting" can be used. (See [34, Section 4.3].) Create a subpopulation of "coin" agents whose only job is to provide random bits to the remaining "main" agents. Main agents build up a list of random bit values to use in the main algorithm, which they obtain when interacting with coin agents. Coin agents start (after first being assigned to the coin subpopulation) in state J, with the following transitions: $J, J \to K, K$; $K, K \to J, J$; $J, K \to C_0, C_1$. When a main agent interacts with C_b, it appends bit b to its list of random bits. Since the above transitions ensure that there are exactly the same number of C_0 and C_1 agents at any time, the bits are unbiased. Since the scheduler ensures that, conditioned on an interaction being between a main and a coin agent, the choice of coin agent is independent of other main-coin interactions, the bit values built up in main agents are independent.[10]

[8] For $n = 2^k, k \in \mathbb{N}$, the population converges to having one unique `active` agent, and all not `active` agents will store the floor of $\log n$. For other values of $n \neq 2^k, k \in \mathbb{N}$, the population converges to $O(\log n)$ `active` agents each having a different value of $\{0, \ldots, \lfloor \log n \rfloor\}$ that results in all null interactions. Note that the interaction between (`active`, 2^3), (`active`, 2^5), concludes with both agents having the same states.

To be concrete, exactly b_1 `active` agents will remain, such that b_1 is the number of 1s in the binary expansion of n. Each of the b_1 `active` agents hold one of the values $i_1, \ldots, i_{b_1}$ for all the indices that have 1 in the binary expansion of n. Thus, to enforce "all" agents (both `active` and not`active`) report the value of $\log n$, the protocol needs at most $O(\log^2 n)$ states per agent.

[9] In more powerful variants of the model, each agent runs a randomized Turing machine [21,23,24]. In this case the $\frac{1}{2}$-geometric random variable can be generated in one step.

[10] The only difference with a truly randomized protocol is that main agents may have to wait to build up random bits before being allowed to do a randomized transition with another agent. This does introduce some dependence in the main protocol, which means this is not a fully black-box technique for replacing a randomized protocol with a deterministic protocol.

Protocol 4 `JuntaElection(rec, sen)`.

Initialization:
`agent.level` $\leftarrow 0$; `agent.active` $\leftarrow$ True

```
1: if rec.active & rec.level = 0 then
2:     rec.level ← 1                                   ▷ happens at the first interaction
3: if sen.active & sen.level = 0 then
4:     sen.active ← False                              ▷ happens at the first interaction
5: else if rec.active & rec.level > 0 then
6:     if rec.level < sen.level then
7:         rec.level ← rec.level + 1
8:     else if rec.level ≥ sen.level then
9:         rec.active ← False
```

4.3. Arbitrary biased coins

One can approximate $\log\log n$ with access to random bits with arbitrary bias. We explained above how to approximate $\log n$ with a series of $\frac{1}{2}$-biased coins. Consider instead a special coin whose initial bias (probability of tail, i.e., continuing to flip) of $\frac{1}{2}$ is squared after each coin flip. In other words, the bias is $\frac{1}{2}$ for the first flip, $\frac{1}{4}$ for the second flip, $\frac{1}{16}$ for the third flip, etc. Similarly to the previous protocol, let each agent independently flip this special coin until a head appears and store the number of consecutive tails. In this process, the fraction of agents who get a tail and continue flipping is approximately squared after each flip, so the maximum stored value among n agents is an approximation of $\log\log n$. The junta election protocol of [10] (also explained in [20]) simulates this process without using any coin flips. We describe the modified version of protocol [10] for simplicity [20]. In this protocol the agents store their current coin number using a `level` variable. Initially, all agents start in state (`level` $= 0$, `active` $=$ True), and eventually, all will set their `active` to False. Agents increase their `level` value via the *asymmetric* transition rules indicated in Protocol 4.

The combination of the first two if statements in 4 acts similarly to the $\frac{1}{2}$-bias coin. About $n/2$ agents participate in their first interaction as receiver and increase their `level` by 1. Intuitively, in Protocol 4, with αn agents (such that $0 < \alpha < 1$) having `level` $\geq i \geq 1$, there will be about $\alpha^2 n$ with `level` $\geq i + 1$.

It is proven that the maximum `level` value in the population (l^*) is an additive approximation of $\log\log n$. More precisely, $\log\log n - 4 \leq l^* \leq \log\log n + 8$ with probability at least $1 - O(1/n)$ [10,20]. This yields a multiplicative factor approximation of $\log n$; see Table 2.

The above protocol is also a so-called *junta election* protocol: the number of agents who obtain the maximum `level` is $O(\sqrt{n \log n})$ with high probability. These agents can be used, for example to create a "junta-driven phase clock" [10], useful for synchronization.

4.4. Fast protocol with additive error

Doty and Eftekhari [21] presented a protocol that improves the approximation factor of the protocol of [6], which approximates $\log n$ using maximum of n geometric random variables, from a multiplicative to an additive factor approximation. Their protocol converges to an estimation of $\log n$ in the interval $[\log n - 5.7, \log n + 5.7]$ after $O(\log^2 n)$ time using $O(\log^4 n)$ number of states per agent. They extend one round of taking the maximum of n geometric random variables of [6] to K rounds of taking maximums and computing their average as an approximation of $\log n$. Doty and Eftekhari [21] proved that the computed average is within $O(1)$ of $\log n$ with $K = \Omega(\log n)$. In their protocol, the agents agree on K by taking the maximum of n independent geometric random variables. For the rest of the protocol, the agents simulate a uniform variation of the leaderless phase clock introduced in [12] to synchronize the agents for $K = O(\log n)$ rounds. In a leaderless phase clock, all agents individually[11] count their number of interactions and compare it with a threshold value $\Theta(\log n)$. If agents' counts reach the threshold, they simply move to the next round of the protocol and set their count value to zero. In each round of the protocol, the agents generate one new geometric random variable, propagate it, and store the maximum. After K rounds, the agents learn K values, each is a maximum of n independent geometric random variables, in sequence and take their sum. In round $K + 1$, the agent divides the sum by K and stores the result as their output.

The protocols we have discussed so far for approximating the population size are leaderless. Their correctness is not dependent on the existence of a unique leader. In a leaderless protocol, all agents are initially equivalent, and there is no distinguished leader.

4.5. Leader-driven, epidemic-based protocol

A leader-driven protocol for approximating the population size was introduced in [2]. The basic idea of their protocol relies on the completion time of an epidemic process. Specifically, the leader triggers an epidemic process (infecting exactly

[11] The leaderless phase clock of the protocol [21] allows agents to increment their counts at every interaction: $c_i, c_j \rightarrow c_{i+1}, c_{j+1}$. In contrast, in a "leaderless phase clock with power of two choices" [12,14] if two agents with counts i and j interact, only the agent with smaller count value increments: $c_i, c_j \rightarrow c_{i+1}, c_j$ for $c_i \leq c_j$. The latter phase clock keeps agents' count values tightly close to each other.

Protocol 5 `PowersOfTwoAveraging(rec, sen)`.

Initialization:
if `agent.leaderBit` = True: `agent.average` $\leftarrow m$
if `agent.leaderBit` = False: `agent.average` $\leftarrow -1$
1: **if** `rec.average` $= -1$ & `sen.average` > 0 **then**
2: `rec.average, sen.average` $\leftarrow$ `sen.average` $- 1$
3: **else if** `sen.average` $= -1$ & `rec.average` > 0 **then**
4: `rec.average, sen.average` $\leftarrow$ `rec.average` $- 1$

one agent L, $Q \to L^*, A$) and keeps track of the number of infected (c_a) and uninfected (c_q) agents without infecting more agents. The followers (initialized in state Q) participate in this protocol via the one-way epidemic rule ($A, Q \to A, A$). As soon as the number of infected and uninfected agents becomes equal, the leader terminates the protocol and reports the approximation $n' = 2^{c_a+1}$. In [2] it was shown that $c_a \leq 2 \log n$ with high probability.[12] This protocol approximates the population size using $\Theta(\log n)$ states for leader while the followers use constant states.

4.6. Discrete averaging with powers of two

Berenbrink, Kaaser, and Radzik [27] introduced a new averaging protocol via modifying the rules of Protocol 1. Recall that Protocol 1 works by pairwise averaging of nonnegative integer values held by each agent; the modified rules restrict the agents to use numbers that are a perfect power of two. The authors carefully developed the protocol such that the new rules of the protocol still preserve the total sum among the agents. In this variation of the averaging protocol, shown in Protocol 5, the agents can store either a perfect power of two or zero. The constraint helps to reduce the space complexity via representing an integer 2^x with x. To show the exact value of 0, the agents use -1. Using a similar approach to [24,27] for the exact counting problem, a leader starts an averaging process with a large (with respect to n) positive value, and all the followers start with zero (`average` $= -1$).

Utilizing the restricted version of the discrete average process, they proposed an approximate counting protocol that outputs the value $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$ with high probability, using $O(\log^2 n)$ time and $O(\log n \log \log n)$ states. To stably solve the approximate counting problem, they used multiple always correct error detection schemes that point the population to the slow token-passing Protocol 3 if an error occurs. With an overhead of $O(\log n)$ states, their protocol stabilizes to $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$ using $O(\log^2 n \log \log n)$ states after $O(\log^2 n)$ time.

For the fast computation of $\log n$, similar to the exact counting protocol of [27] described in Section 3, all the agents simulate the junta election protocol of [10] at the very beginning to (1) simulate a junta driven phase clock and achieve synchronization and (2) elect a unique leader for the discrete averaging Protocol 5. Once the population elected its leader, the leader performs a linear search, starts 0, to find $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$. At the beginning of round i, the agents reset their `average` back to -1 and follow the rules of Protocol 5 while the leader starts with `average` $= i$ (injecting 2^i tokens to the population). At the end of round i, if some agents hold an average value greater than zero (`average` ≥ 1), the leader will stop the search and broadcast the value i as an approximation of $\log n$. The author proved that w.h.p. the leader stops the search after $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$ rounds.

4.7. Regulating size in the presence of an adversary

Goldwasser, Ostrovsky, Scafuro, and Sealfon [42] studied a variation of population protocols, which allows agents individually to decide to replicate or self-destruct, changing the population size in response to an adversary who can add or remove arbitrary number of agents. They proposed a protocol that can approximately maintain a target population size (both the initial population size and the target are assumed to be known to each agent) despite this adversary.

However, they use a *synchronous* variation of population protocols: in one round of the computation, a constant fraction of agents interact (at most once) via a random matching of size $k = O(n)$. Observe that, unlike the asynchronous scheduler, the synchronous scheduler prevents any agent having multiple interactions per k total interactions. Despite this difference in definitions, it is conceivable that techniques used in the analysis of [42] could be applied to the standard population protocol model. It is also noteworthy that their model of agents being created and destroyed is expressible in the model of chemical reaction networks [43], of which population protocols are a special case.

5. Tools for making nonuniform protocols uniform

Part of the practical motivation behind the study of the counting problem comes from the existence of nonuniform protocols and a desire to create uniform variants of them. Since most nonuniform protocols require advance knowledge of $\log n$, the basic technique for making such a protocol uniform is to first compute an estimate of $\log n$ using a protocol from Section 4, then to compose this with the existing nonuniform protocol, replacing its estimate of $\log n$ with this computed

[12] Theorem 1 of [2] states that $\log n \leq c_a$ with high probability. However, this does not appear to hold in simulation. It seems likely that a bound of $c \cdot \log n$ can be proven for some $c > 0$ based on known results lower bounding times for epidemics to spread [41].

value. We break this section into two pieces. First, we recall a few size approximation protocols from Section 4, comparing them in accuracy, space, and simplicity with specific suggestions for how to account for these properties in choosing one to be composed with a nonuniform protocol. In the next part, assuming we have a protocol that computes an estimate of the population size, we show how to compose it with a nonuniform protocol.

5.1. Comparison of approximate counting protocols

In this part, we compare techniques that solve the approximate counting problem. See Table 2 for a detailed comparison. For most nonuniform protocols (e.g., a leaderless phase clock [12]), a value that is $\Theta(\log n)$ suffices. Thus we lead the discussion with protocols that approximate $\log n$ within a multiplicative factor error. Although additive approximate error is often unnecessary, in some circumstances, one may require the estimate be to exactly $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$. For example, the uniform majority protocol of [18] requires the estimate to be at least $\lceil \log n \rceil$ with probability 1.

Simple 2-approximation [Section 4.2] Taking the maximum of n geometric random variables provides a 2-factor approximation of $\log n$ [6,21,40] with probability at least $1 - O(1/n)$,[13] and takes $O(\log n)$ time to converge. Although this approach is very simple and straightforward for composition, the space complexity of the protocol is not bounded with probability 1, and the agents use $O(\log n)$ states w.h.p.

Minimal space overhead [Section 4.3] Recall that the maximum level l^* in the junta election protocol is $\lfloor \log \log n \rfloor - 3 \leq l^* \leq \log \log n + 4(a+1)$ with probability at least $1 - 1/n^a$ [10,20]. Despite the large multiplicative approximation factor for computing $\lceil \log n \rceil$, the junta election protocol [10] imposes minimal, $O(\log \log n)$, space overhead and converges in $O(\log n)$ time. Moreover, the protocol provides a junta of n^ϵ for $0 \leq \epsilon < 1$ leaders that can simulate a "junta-driven phase clock" to synchronize agents in phases of length $\Theta(\log n)$ time [10]. See [1,34] for more details about the phase clock.

Maximizing accuracy, always correct Two protocols from sections 4.1, 4.6 compute $\lfloor \log n \rfloor$ (or $\lceil \log n \rceil$). Both protocols provide probability-1 correctness using $O(n \log n)$ and $O(\log^2 n)$ time respectively. The former is much simpler and is used as a "slow backup" subroutine in the $O(\log n)$ time protocol of [18]. Although it is much slower than $O(\log n)$ time, since it is needed only with low probability, it contributes negligibly to the expected time.

5.2. Composition of an uniform counting protocol with a nonuniform protocol

Most of the time, we can construct a uniform protocol from a nonuniform protocol through composition with a uniform approximate counting protocol. Even though we are unaware of any black-box theorem that proves the correctness of the restarting technique under any circumstance, the authors of [10,18,21,24,27] used the procedure discussed below and proved it correct with an ad-hoc analysis. We explain in a general way how to use these approximate counting protocols to make a nonuniform protocol uniform in the next part.

Note that all of the approximate counting protocols mentioned in Section 4, except approximating with a leader explained in Section 4.5, are not terminating. In other words, the agents are not aware of the completion of the protocol. Although termination is impossible in the uniform model of population protocols [21], we can try composing two protocols without using termination of the upstream ones. For a concrete discussion, consider protocols **U** and **D** such that **U** is a uniform approximate counting and **D** is a general nonuniform protocol; **U** is the *upstream* protocol whose output is given as input to the *downstream* protocol **D**. To construct a uniform protocol, we summarize a simple restarting technique that has been used widely [10,24,27] to compose protocols **U** and **D**. In this technique, we run both protocols in parallel in the population. If the `count` fields of the protocol **U** in the agents' memory change, then a signal will be propagated (by epidemic) through the population to notify all agents with the updated `count`. This signal will stop the protocol **D** (or parts of protocol **D** that are dependent on the population size) and reinitialize it with the updated `count`, which eventually will be an approximation of $\log n$.

Despite the difference between the initial configuration of a nonuniform protocol and the one after the restart signal (agents do not have the same state), any agents who participate in the last execution of protocol **D** restarts their memory (related fields concerning protocol **D**) to the initial values. Thus, a high probability correct counting scheme can also guarantee the correctness of the downstream protocol **D** w.h.p.

6. Conclusion and open questions

In this paper, we gave a brief description of existing protocols that exactly or approximately compute the population size n. We also discussed a technique for converting nonuniform protocols (those that assume agents are initialized with an approximate estimate of n) to uniform protocols, by composing size approximation with the nonuniform protocol.

While our focus in this paper is the size counting problem, the mentioned protocols demonstrate general techniques that can help solve other problems and design new protocols. Alistarh and Gelashvilli [34] mentioned different ideas such

[13] In fact the lower bound is stronger: the maximum is between $\log_2 n - \log_2(n) \cdot \ln n$ and $2 \cdot \log_2 n$ with probability at least $1 - O(1/n)$; see [21, Lemma 3.8].

as the space multiplexing used in [18,20,21,27] and the junta-driven phase clock [10] as available building blocks to design new protocols. We also summarized two variations of the discrete averaging technique introduced in [31], tightly analyzed in [17,27–29,33], that has since been widely deployed in other protocols to solve the counting problem [17,24,27,32] and the exact majority problem [18,31,32]. (See Protocols 1 and 5.)

Open questions with composition of two uniform protocols We discussed how to make a nonuniform protocol uniform through composition with a uniform counting protocol that allows the nonuniform protocol to use the output of the counting protocol. Generally, in a composition of a uniform protocol **U** with a protocol **D**, protocol **D** might get restarted repeatedly. In each restart, the agents propagate a new signal with updated information about the size. The counting protocol might even generate a new restart signal before the previous signal hits all the agents. Having this in mind, if a protocol uses duplicate restart signals, restarted and deprecated agents could become indistinguishable. For example, using restart signals of constant size might create inconsistency in the population.

Unique (and perhaps monotonically increasing) restarting signals guarantees the correctness of the downstream protocol. Since eventually, all agents agree on the last (largest) restart signal and restart protocol **D** for the final execution. Even assuming a monotone increasing restart signal might change some probability bounds on the convergence time of protocols. The current literature lacks a general-purpose theorem that proves under what conditions of a downstream protocol the restarting technique works.

Collective output representation of the population size Moreover, all the counting protocols summarized in this paper require all agents eventually to represent the computed count. If agents are required to store the value n, then there is clearly a linear-state lower bound, since $\log n$ bits are required merely to write n. However, what if no agent individually stores all of n? Consider instead a *collective* representation of the population size, where some agents each store (for example) one bit of n, as well as the significance of the bit.

With this trick, the lower bound does not apply anymore. There might exist a protocol that solves the exact counting with $o(n)$ states. However, readout could be more difficult, since we cannot simply look at the memory of a single agent and read the population size; instead, we must sample a small subset of the population. For example, composing size computation with another protocol would be less straightforward since the agents who are computing the downstream protocol would not at any point have access to all the bits of n. One could imagine a protocol that spreads the output to the whole population almost equally: for example, having $O(n/\log n)$ agents responsible for each index of the binary expansion of n. With this trick, the output will be present dense enough among the population. Thus, a random sample of polylogarithmic agents would have enough information to reconstruct the value of n.

The $O(n \log n)$ time slow size approximation protocol (Protocol 3) collectively represents n: each remaining active agent holds a value k such that there is a 1 at significance k in the binary expansion of n. Is there a sublinear-time, sublinear-state protocol, so that the agents report the population size via this collective representation? A valid solution to the exact counting problem with a collective output also solves the *parity* problem: compute the least significant bit of n.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] D. Angluin, J. Aspnes, D. Eisenstat, Fast computation by population protocols with a leader, Distrib. Comput. 21 (3) (2008) 183–199.
[2] O. Michail, P.G. Spirakis, M. Theofilatos, Simple and fast approximate counting and leader election in populations, Inf. Comput. (2021) 104698, https://doi.org/10.1016/j.ic.2021.104698, https://www.sciencedirect.com/science/article/pii/S0890540121000134.
[3] H.-L. Chen, D. Doty, D. Soloveichik, Deterministic function computation with chemical reaction networks, Nat. Comput. 13 (4) (2014) 517–534, https://doi.org/10.1007/s11047-013-9393-6, special issue of invited papers from DNA 2012.
[4] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, Distrib. Comput. 18 (4) (2006) 235–253.
[5] D. Doty, D. Soloveichik, Stable leader election in population protocols requires linear time, Distrib. Comput. 31 (4) (2018) 257–271, special issue of DISC 2015, invited papers.
[6] D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, R.L. Rivest, Time-space trade-offs in population protocols, in: SODA 2017: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2017, pp. 2560–2579.
[7] A. Belleville, D. Doty, D. Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, in: ICALP 2017: 44th International Colloquium on Automata, Languages, and Programming, in: LIPIcs, vol. 80, 2017, pp. 141:1–141:14.
[8] D. Alistarh, R. Gelashvili, Polylogarithmic-time leader election in population protocols, in: Proceedings, Part II, of the 42nd International Colloquium on Automata, Languages, and Programming, vol. 9135, ICALP 2015, Springer-Verlag, 2015, pp. 479–491.
[9] P. Berenbrink, D. Kaaser, P. Kling, L. Otterbach, Simple and efficient leader election, in: 1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA, 2018, pp. 9:1–9:11.
[10] L. Gąsieniec, G. Stachowiak, Fast space optimal leader election in population protocols, in: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18, Society for Industrial and Applied Mathematics, USA, 2018, pp. 265–266.
[11] L. Gąsieniec, G. Stachowiak, P. Uznański, Almost logarithmic-time space optimal leader election in population protocols, in: The 31st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '19, Association for Computing Machinery, 2019, pp. 93–102.

[12] D. Alistarh, J. Aspnes, R. Gelashvili, Space-optimal majority in population protocols, in: SODA 2018: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2018, pp. 2221–2239.
[13] P. Berenbrink, R. Elsässer, T. Friedetzky, D. Kaaser, P. Kling, T. Radzik, A Population Protocol for Exact Majority with $O(\log^{5/3} n)$ Stabilization Time and Theta(log n) States, in: U. Schmid, J. Widder (Eds.), 32nd International Symposium on Distributed Computing, DISC 2018, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 121, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018, pp. 10:1–10:18, http://drops.dagstuhl.de/opus/volltexte/2018/9799.
[14] S. Ben-Nun, T. Kopelowitz, M. Kraus, E. Porat, An $O(\log^{3/2} n)$ parallel time population protocol for majority with $O(\log n)$ states, in: Proceedings of the 39th Symposium on Principles of Distributed Computing, PODC '20, Association for Computing Machinery, 2020, pp. 191–199.
[15] Y. Sudo, F. Ooshita, T. Izumi, H. Kakugawa, T. Masuzawa, Logarithmic expected-time leader election in population protocol model, in: M. Ghaffari, M. Nesterenko, S. Tixeuil, S. Tucci, Y. Yamauchi (Eds.), Stabilization, Safety, and Security of Distributed Systems, Springer International Publishing, Cham, 2019, pp. 323–337.
[16] A. Bilke, C. Cooper, R. Elsässer, T. 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: Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC '17, Association for Computing Machinery, 2017, pp. 451–453.
[17] Y. Mocquard, E. Anceaume, J. Aspnes, Y. Busnel, B. Sericola, Counting with population protocols, in: 14th IEEE International Symposium on Network Computing and Applications, 2015, pp. 35–42.
[18] D. Doty, M. Eftekhari, L. Gąsieniec, E. Severson, G. Stachowiak, P. Uznański, A time and space optimal stable population protocol solving exact majority, arXiv:2106.10201, 2021.
[19] P. Berenbrink, G. Giakkoupis, P. Kling, Optimal time and space leader election in population protocols, in: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Association for Computing Machinery, New York, NY, USA, 2020, pp. 119–129.
[20] P. Berenbrink, R. Elsässer, T. Friedetzky, D. Kaaser, P. Kling, T. Radzik, Time-space trade-offs in population protocols for the majority problem, Distributed Computing, https://doi.org/10.1007/s00446-020-00385-0.
[21] D. Doty, M. Eftekhari, Efficient size estimation and impossibility of termination in uniform dense population protocols, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19, Association for Computing Machinery, 2019, pp. 34–42.
[22] D. Angluin, J. Aspnes, D. Eisenstat, Stably computable predicates are semilinear, in: 25th annual ACM Symposium on Principles of Distributed Computing, PODC, ACM Press, 2006, pp. 292–299.
[23] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, P.G. Spirakis, Passively mobile communicating machines that use restricted space, Theor. Comput. Sci. 412 (46) (2011) 6469–6483.
[24] D. Doty, M. Eftekhari, O. Michail, P.G. Spirakis, M. Theofilatos, Brief announcement: Exact size counting in uniform population protocols in nearly logarithmic time, in: U. Schmid, J. Widder (Eds.), 32nd International Symposium on Distributed Computing, DISC 2018, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 121, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018, pp. 46:1–46:3, http://drops.dagstuhl.de/opus/volltexte/2018/9835.
[25] D.T. Gillespie, Exact stochastic simulation of coupled chemical reactions, J. Phys. Chem. 81 (25) (1977) 2340–2361.
[26] O. Bournez, J. Chalopin, J. Cohen, X. Koegler, M. Rabie, Population protocols that correspond to symmetric games, Int. J. Unconv. Comput. 9 (2013) 5–36.
[27] P. Berenbrink, D. Kaaser, T. Radzik, On counting the population size, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19, Association for Computing Machinery, 2019, pp. 43–52.
[28] P. Berenbrink, T. Friedetzky, D. Kaaser, P. Kling, Tight and simple load balancing, in: 2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS, 2019, pp. 718–726.
[29] P. Berenbrink, R. Klasing, A. Kosowski, F. Mallmann-Trenn, P. Uznański, Improved analysis of deterministic load-balancing schemes, ACM Trans. Algorithms 15 (1), https://doi.org/10.1145/3282435.
[30] T. Sauerwald, H. Sun, Tight bounds for randomized load balancing on arbitrary network topologies, in: 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, IEEE, 2012, pp. 341–350.
[31] D. Alistarh, R. Gelashvili, M. Vojnović, Fast and exact majority in population protocols, in: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 47–56.
[32] Y. Mocquard, E. Anceaume, B. Sericola, Optimal proportion computation with population protocols, in: 2016 IEEE 15th International Symposium on Network Computing and Applications, NCA, 2016, pp. 216–223.
[33] Y. Mocquard, B. Sericola, E. Anceaume, Explicit and tight bounds of the convergence time of average-based population protocols, in: International Colloquium on Structural Information and Communication Complexity, Springer, 2019, pp. 357–360.
[34] D. Alistarh, R. Gelashvili, Recent algorithmic advances in population protocols, SIGACT News 49 (3) (2018) 63–73, https://doi.org/10.1145/3289137.3289150.
[35] T. Amir, J. Aspnes, D. Doty, M. Eftekhari, E. Severson, Message complexity of population protocols, in: H. Attiya (Ed.), 34th International Symposium on Distributed Computing, DISC 2020, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 179, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 6:1–6:18, https://drops.dagstuhl.de/opus/volltexte/2020/13084.
[36] J. Aspnes, J. Beauquier, J. Burman, D. Sohier, Time and space optimal counting in population protocols, in: 20th International Conference on Principles of Distributed Systems (OPODIS 2016), vol. 70, 2017, pp. 13:1–13:17.
[37] J. Beauquier, J. Burman, S. Clavière, D. Sohier, Space-optimal counting in population protocols, in: Y. Moses (Ed.), Distributed Computing, Springer, Berlin, Heidelberg, 2015, pp. 631–646.
[38] J. Beauquier, J. Clement, S. Messika, L. Rosaz, B. Rozoy, Self-stabilizing counting in mobile sensor networks with a base station, in: A. Pelc (Ed.), Distributed Computing, Springer, Berlin, Heidelberg, 2007, pp. 63–76.
[39] T. Izumi, K. Kinpara, T. Izumi, K. Wada, Space-efficient self-stabilizing counting population protocols on mobile sensor networks, Theor. Comput. Sci. 552 (2014) 99–108, https://doi.org/10.1016/j.tcs.2014.07.028, https://www.sciencedirect.com/science/article/pii/S0304397514005970.
[40] B. Eisenberg, On the expectation of the maximum of IID geometric random variables, Stat. Probab. Lett. 78 (2) (2008) 135–143, https://doi.org/10.1016/j.spl.2007.05.011, http://www.sciencedirect.com/science/article/pii/S0167715207002040.
[41] Y. Mocquard, B. Sericola, S. Robert, E. Anceaume, Analysis of the propagation time of a rumour in large-scale distributed systems, in: 2016 IEEE 15th International Symposium on Network Computing and Applications, NCA, 2016, pp. 264–271.
[42] S. Goldwasser, R. Ostrovsky, A. Scafuro, A. Sealfon, Population stability: regulating size in the presence of an adversary, in: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, ACM, 2018, pp. 397–406.
[43] D. Soloveichik, M. Cook, E. Winfree, J. Bruck, Computation with finite stochastic chemical reaction networks, Nat. Comput. 7 (4) (2008) 615–633, https://doi.org/10.1007/s11047-008-9067-y.