Dynamic Size Counting in Population Protocols

David Doty
University of California, Davis, CA, USA

Mahsa Eftekhari
University of California, Davis, CA, USA

Abstract

The population protocol model describes a network of anonymous agents that interact asynchronously in pairs chosen at random. Each agent starts in the same initial state s. We introduce the *dynamic size counting* problem: approximately counting the number of agents in the presence of an adversary who at any time can remove any number of agents or add any number of new agents in state s. A valid solution requires that after each addition/removal event, resulting in population size n, with high probability each agent "quickly" computes the same constant-factor estimate of the value $\log_2 n$ (how quickly is called the *convergence* time), which remains the output of every agent for as long as possible (the *holding* time). Since the adversary can remove agents, the holding time is necessarily finite: even after the adversary stops altering the population, it is impossible to *stabilize* to an output that never again changes.

We first show that a protocol solves the dynamic size counting problem if and only if it solves the *loosely-stabilizing counting* problem: that of estimating $\log n$ in a *fixed-size* population, but where the adversary can initialize each agent in an arbitrary state, with the same convergence time and holding time. We then show a protocol solving the loosely-stabilizing counting problem with the following guarantees: if the population size is n, M is the largest initial estimate of $\log n$, and s is the maximum integer initially stored in any field of the agents' memory, we have expected convergence time $O(\log n + \log M)$, expected polynomial holding time, and expected memory usage of $O(\log^2(s) + (\log\log n)^2)$ bits. Interpreted as a dynamic size counting protocol, when changing from population size n_{prev} to n_{next}, the convergence time is $O(\log n_{\text{next}} + \log\log n_{\text{prev}})$.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Models of computation

Keywords and phrases Loosely-stabilizing, population protocols, size counting

Digital Object Identifier 10.4230/LIPIcs.SAND.2022.13

Related Version *Full Version*: `https://arxiv.org/abs/2202.12864`

Supplementary Material *Software (Simulation Results with Colab Notebook)*:
`https://github.com/eftekhari-mhs/population-protocols/tree/main/dynamic_counting`
archived at `swh:1:dir:a71288ec3836738d716285e3e6f6446978940c2f`

Funding Supported by NSF award 1900931 and CAREER award 1844976.

1 Introduction

A population protocol [6] is a network of n anonymous and identical *agents* with finite memory called the *state*. A scheduler repeatedly selects a pair of agents independently and uniformly at random to interact. Each agent sees the entire state of the other agent in the interaction and updates own state in response. Time complexity is measured by *parallel time*: the number of interactions divided by the population size n, capturing the natural time scale in which each agent has $\Theta(1)$ interactions per unit time. The agents collectively do a computation, e.g., population size counting: computing the value n. Counting is a fundamental task in distributed computing: knowing an estimate of n often simplifies the design of protocols solving problems such as majority and leader election [1, 2, 4, 11, 13–16, 24, 31, 35].


© David Doty and Mahsa Eftekhari;
licensed under Creative Commons License CC-BY 4.0

A protocol is defined by a *transition function* with a pair of states as input and as output (more generally to capture randomized protocols, a relation that can associate multiple outputs to the same input). For example, consider the simple counting protocol with transitions $L_i, L_j \to L_{i+j}, F_{i+j}$, with every agent starting in L_1. In population size n, this protocol converges to a single agent in state L_n, with all other agents in state F_i for some i. The additional transitions $F_i, F_j \to F_j, F_j$ for $i < j$ propagate the output n to all agents.

The dynamic size counting problem

In contrast to most work, which assumes the population size n is fixed over time, we model an adversary that can add or remove agents arbitrarily and repeatedly during the computation. All agents start in the same state, including newly added agents. The goal is for each agent to approximately count the population size n, which we define to mean that all agents should eventually store the same output k in their states, which with high probability is within a constant multiplicative factor of $\log n$.[1] Once all agents have the same output k, they have *converged*. They maintain k as the output for some time called the *holding time* (after which they might alter k even if the population size has not changed). In response to a "significant" change in size from n_{prev} to n_{next}, agents should re-converge to a new output k' of $\log n_{\text{next}}$. (Agents are not "notified" about the change; instead they must continually monitor the population to test whether their current output is accurate.) Note that if n_{prev} is close to n_{next} (within a polynomial factor), then k may remain an accurate estimate of n_{next}, so agents may not re-converge in response to a small change.

Ideally the expected convergence time is small, and the expected holding time is large. With a fixed size population, it is common to require the output to *stabilize* to a value that never again changes after convergence, i.e., infinite holding time. However, this turns out to be impossible with an adversary that can remove agents (Observation 3.4). When changing from size n_{prev} to n_{next}, our protocol achieves expected convergence time $O(\log n_{\text{next}} + \log\log n_{\text{prev}})$ and expected holding time $\Omega(n_{\text{next}}^c)$, where c can be made arbitrarily large. The number of bits of memory used per agent is $O(\log^2(s) + (\log\log n)^2)$, where s is the maximum integer stored in the agents' memory after the change.

While it is common to measure population protocol memory complexity by counting the number of states (which is exponentially larger than the number of bits required to represent each state), that measure is a bit awkward here. Our protocol is uniform – the same transition rules for every population size – so has an infinite number of producible states. One could count expected number of states that will be produced, but this is a bit misleading: in time t each agent visits $O(t)$ states on average, so $O(t \cdot n)$ states total. Counting how many bits are required is more accurate metric of the actual memory requirements.

The loosely-stabilizing counting problem

The dynamic size counting problem has an equivalent characterization: rather than removing agents and adding them with a fixed initial state, the *loosely-stabilizing* adversary sets each agent to an arbitrary initial state in a fixed-size population. A protocol solves the dynamic size counting problem if and only if it solves the loosely-stabilizing counting problem, with the same convergence and holding times (Lemma 3.5). Due to this equivalence, we analyze

[1] *Nonuniform* protocols require agents to be initialized with an estimate k of $\log n$ in order to accomplish other tasks, such as a "leaderless phase clock" [1]. The bound $k = \Theta(\log n)$ is necessary and sufficient for correctness and speed in most cases [1, 2, 4, 11, 13–16, 24, 31, 35].

our protocol assuming a fixed population size and adversarial initial states. In this case our convergence time $O(\log n + \log M)$ is measured as a function of the population size n and the value M that is the maximum `estimate` value stored in agents' memory. From the perspective of the dynamic size counting problem, these "adversarial initial states" would correspond to the agent states after correctly estimating the *previous* population size, just prior to adding or removing agents.

1.1 Related work

Initialized counting with a fixed size population. In population protocols with fixed size, there is work computing exactly or approximately the population size n. For a full review see [19]. Such protocols reach a *stable* configuration from which the output cannot change. Some of these counting protocols would still solve the counting problem in the presence of an adversary who can only add agents (see Observation 3.3). However, these protocols fail in the presence of an adversary who can also remove agents, since they work only in the initialized setting and rely on reaching a stable configuration (see Observation 3.4).

Self-stabilizing counting with a fixed size population. A population protocol is *self-stabilizing* if, from *any* initial configuration, it reaches to a correct stable configuration. Self-stabilizing size counting has been studied [8–10, 27], but provably requires adding a "base station" agent that cannot be corrupted by the adversary. In these protocols the base station is the only agent required to learn the population size. Aspnes, Beauquier, Burman, and Sohier [8] showed a time- and space-optimal protocol that solves the exact counting problem in $O(n \log n)$ time, using 1-bit memory for each non-base station agent.

Size regulation in a dynamically sized population. The model described by Goldwasser, Ostrovsky, Scafuro, and Sealfon [25] is close to our setting. They consider the *size regulation problem*: approximately maintaining a target size (hard-coded into each agent) using $O(\log \log n)$ bits of memory per agent, despite an adversary that (like ours) adds or removes agents. That paper assumes a model variation in which:

- The agents can replicate or self-destruct.
- The computation happens through synchronized rounds of interactions. At each round the scheduler selects a random matching of size $k = O(n)$ agents to interact.
- The adversary's changes to the population size are limited. The adversary can insert or delete a total of $o(n^{1/4})$ agents within each round.

The latter two model differences above crucially rule out their protocol as useful for our problem. We use the standard asynchronous scheduler, and much of the complexity of our protocol is to handle drastic population size changes (e.g., removing $n - \log n$ agents). Additionally, their protocol heavily relies on flipping coins of bias $\frac{1}{\sqrt{n}}$ that we cannot utilize since the agents don't start with an estimate of n. Moreover, even when the agents compute their estimate, the population size might change.

Loosely-stabilizing leader election. Sudo, Nakamura, Yamauchi, Ooshita, Kakugawa, and Masuzawa [34] introduce loose-stabilization as a relaxation for the self-stabilizing leader election problem in which the agents must know the exact population size to elect a leader. The loosely stabilizing leader election guarantees that starting from any configuration, the population will elect a leader within a short time. After that, the agents hold the leader for a long time but not forever (in contrast with self-stabilization). On the positive side, the agents no longer need to know the exact population size to solve the loosely-stabilizing

leader election, but a rough upper bound suffices. Loosely-stabilizing leader election has been studied, providing a time-optimal protocol that solves the leader election problem [33] and a tradeoff between the holding and convergence times [26, 36].

Computation with dynamically changing inputs. Alistarh, Töpfer, and Uznański [5] consider the dynamic variant of the comparison problem. In the comparision problem, a subset of population are in the input states X and Y and the goal is to compute if $X > Y$ or $X < Y$. In the dynamic variant of the comparision problem, they assume an adversary who can change the counts of the input states at any time. The agents should compute the output as long as the counts remain untouched for sufficiently long time. They propose a protocol that solves the comparision problem in $O(\log n)$ time using $O(\log n)$ states per agent, assuming $|X| \geq C_2 \cdot |Y| \geq C_1 \log n$ for some constants $C_1, C_2 > 1$.

Berenbrink, Biermeier, Hahn, and Kaaser [12] consider the adaptive majority problem (generalization of the comparison problem [5]). At any time every agent has an opinion from $\{X, Y\}$ or undecided and their opinions might change adversarially. The goal is to have agreement in the population about the majority opinion. They introduce a non-uniform loosely-stabilizing leaderless phase clock that that uses $O(\log n)$ states to solve the adaptive majority problem. This is similar to having an adversary who can add or remove agents with different opinion. However, all agents are assumed already to have an estimate of $\log n$ that remains untouched. Thus it is not straightforward to use their protocol to solve our problem of obtaining this estimate.

2 Definitions and Notation

A *population protocol* is a pair $\mathcal{P} = (\Lambda, \Delta)$, where Λ is a finite set of *states*, and $\Delta \subseteq (\Lambda \times \Lambda) \times (\Lambda \times \Lambda)$ is the *transition relation*. (Often this is defined as a function $\delta : \Lambda \times \Lambda \to \Lambda \times \Lambda$, but we allow randomized transitions, where the same pair of inputs can randomly choose among multiple outputs.)

A *configuration* $\mathbf{c}$ of a population protocol is a multiset over Λ of size n, giving the states of the n agents in the population. For a state $s \in \Lambda$, we write $\mathbf{c}(s)$ to denote the count of agents in state s. A *transition* is a 4-tuple, written $\alpha : r_1, r_2 \to p_1, p_2$, such that $((r_1, r_2), (p_1, p_2)) \in \Delta$. If an agent in state r_1 interacts with an agent in state r_2, then they can change states to p_1 and p_2. This notation omits explicit probabilities; our main protocol's transitions can be implemented so as to always have either one or two possible outputs for any input pair, with probability $1/2$ of each output in the latter case.[2] For every pair of states r_1, r_2 without an explicitly listed transition $r_1, r_2 \to p_1, p_2$, there is an implicit *null* transition $r_1, r_2 \to r_1, r_2$ in which the agents interact but do not change state. For our main protocol, we specify transitions formally with pseudocode that indicate how agents alter each independent field in their state. We say a configuration $\mathbf{d}$ is *reachable* from a configuration $\mathbf{c}$ if applying 0 or more transitions to $\mathbf{c}$ results in $\mathbf{d}$.

When discussing random events in a protocol of population size n, we say event E happens *with high probability* if $\Pr[\neg E] = O(n^{-c})$, where c is a constant that depends on our choice of parameters in the protocol, where c can be made arbitrarily large by changing the parameters.

[2] For the purpose of representation, we make an exception in our protocol, when we show agents generate a geometric random variable in one line (see Protocol 6). However, we can assume a geometric random variable is generated through $O(\log n)$ consecutive interactions with each selecting out of two possible outputs (**H** or **T**).

For concreteness, we will write a particular polynomial probability such as $O(n^{-2})$, but in each case we could tune some parameter (say, increasing the time complexity by a constant factor) to increase the polynomial's exponent.

To measure time we count the total number of interactions (including null transitions such as $a, b \to a, b$ in which the agents interact but do not change state), and divide by the number of agents n.

In a *uniform* protocol (such as the main one of this paper), the transitions are independent from the population size n (see [21] for a formal definition). In other words, a single protocol computes the output correctly when applied on any population size. In contrast, in a nonuniform protocol different transitions are applied for different population sizes.

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*. A population protocol is self-stabilizing if from any initial configuration, the agents stably solve the problem.

3 Dynamic Size Counting

In a population of size n, define $C(n, \epsilon_1, \epsilon_2)$ to be the set of correct configurations $\mathbf{c}$ such that every agent u in $\mathbf{c}$ obeys $\epsilon_1 \log n < u.\texttt{estimate} < \epsilon_2 \log n$. Let t_h be any time bound. Moreover, we define $L(n, t_h) \subset C(n, \epsilon_1, \epsilon_2)$ the subset of correct configurations such that as the expected time for protocol P starting from a configuration $\mathbf{l} \in L(n, t_h)$ to stay in $C(n, \epsilon_1, \epsilon_2)$ is at least $t_h(n)$.

▶ **Definition 3.1.** *Let n_{prev} and n_{next} denote the previous and next population size. A protocol P solves the* dynamic size counting *problem if there are $\epsilon_1, \epsilon_2 > 0$, called the* accuracy*, such that if the population size changes from n_{prev} to n_{next}, the protocol reaches a configuration $\mathbf{l}$ in $L(n_{next}, t_h)$ with high probability. The time needed to do this is called the* convergence *time. Moreover, t_h, the time that the population stays in $C(n_{\text{next}}, \epsilon_1, \epsilon_2)$, is called the* holding *time.*

A population protocol is $(t_c(n), t_h(n))$-*loosely stabilizing* if starting from any initial configuration, the agents reach a correct configuration in $t_c(n)$ time and stay in the correct configuration for additional $t_h(n)$ time [33,34]. In contrast to self-stabilizing [7,17], subsequent interactions can move the agents to an incorrect configuration; however, the agents recover quickly from an incorrect configuration.

Given any starting configuration $\mathbf{s} \notin C(n, \epsilon_1, \epsilon_2)$ of size n, we define $f_c(\mathbf{s}, L(n, t_h))$ as the expected time to reach a correct configuration in $L(n, t_h)$.

▶ **Definition 3.2** ([34, Definition 2])**.** *Let $t_c(n, M)$ and $t_h(n)$ be functions of n, the largest integer value M in the initial configuration $\mathbf{s}$, and the set of correct configuration $C(n, \epsilon_1, \epsilon_2)$. A protocol P is a $t_c(n, M), t_h(n), \epsilon_1, \epsilon_2$ loosely-stabilizing population size counting protocol if there exists a set $L(n, t_h) \subset C(n, \epsilon_1, \epsilon_2)$ of configurations satisfying:*

For every n and every initial configuration $\mathbf{s} \notin C(n, \epsilon_1, \epsilon_2)$ of size n, $f_c(\mathbf{s}, L(n, t_h)) \leq t_c(n, M)$.

3.1 Basic properties of the dynamic size counting problem

We first observe that the key challenge in dynamic size counting is that the adversary may remove agents. If the adversary can only add agents, the problem is straightforward to solve with optimal convergence and holding times.

▶ **Observation 3.3.** *Suppose the adversary in the dynamic size counting problem only adds agents. Then there is a protocol solving dynamic size counting with $O(\log n)$ convergence time (in expectation and with probability $\geq 1 - O(1/n)$) and infinite holding time.*

Proof. Each agent in the initial state s generates a geometric random variable. After the last time that the adversary adds agents, resulting in n total agents, exactly n geometric random variables will have been generated. Agents propagate the maximum by epidemic using transition $a, b \to \max(a, b), \max(a, b)$, taking $3 \ln n$ time to reach all agents with probability $\geq 1 - \frac{1}{n^2}$ [17, Corollary 2.8]. The maximum of n i.i.d. geometric random variables is in the range $[\log n - \log \ln n, 2 \log n]$ with probability $\geq 1 - \frac{1}{n}$ [18, Lemma D.7]. ◀

In contrast, if the adversary can *remove* agents, then even if it is guaranteed to do this exactly once, no protocol can be stabilizing, i.e., have infinite holding time.

▶ **Observation 3.4.** *Suppose the adversary in the dynamic size counting problem will remove agents exactly once. Then any protocol solving the problem has finite holding time.*

Proof. Suppose otherwise. Let the initial population size be n and the later size be $n' < n$. The protocol must handle the case where the adversary *never* removes agents, since in population size n this is equivalent to an adversary who starts with $n + 1$ agents and immediately removes one of them. Thus if the adversary waits sufficiently long before the removal, then all agents stabilize to output $k = \Theta(\log n)$. In other words, no sequence of transitions can alter the value, including transitions occurring only among any subpopulation of size n'. So after the adversary removes $n - n'$ agents, the remaining n' agents are unable to alter the output k, a contradiction if n' is sufficiently small compared to n such that the output k is not a correct estimate for a population of size n'. ◀

Recall that we define M as the largest integer value the agents stored in the starting configuration $\mathbf{s}$. Lemma 3.5 shows that the dynamic size counting problem is equivalent to the loosely-stabilizing counting problem. Due to this equivalence, our correctness proofs will use the loosely-stabilizing characterization. The proof is given in the full version [20].

▶ **Lemma 3.5.** *A protocol solves the dynamic size counting problem with convergence time $t_c(n, M)$ and holding time $t_h(n)$ if and only if it solves the loosely-stabilizing counting problem with convergence time $t_c(n, M)$ and holding time $t_h(n)$.*

Proof sketch. Any states present in an adversarially prepared configuration $\mathbf{c}$ will be produced in large quantities from any sufficiently large initial configuration of all initialized states s [18, Lemma 4.2]. The dynamic size adversary can then remove agents to result in $\mathbf{c}$, which the protocol must handle, showing it can handle an arbitrary initial configuration. ◀

3.2 High-level overview of dynamic size counting protocol

This section briefly describes our protocol for solving the dynamic size counting, defined formally in Section 3.3. By Lemma 3.5, it suffices to design a protocol solving the loosely-stabilizing counting problem for a fixed population size n. Our protocol uses the "detection" protocol of [3]. Consider a subset of states designated as a "source". A detection protocol alerts all agents whether a source state is present in the population.

In Protocol 1, the population maintains several dynamic *groups*, with the agent's group stored as a positive integer field `group`. The `group` values are not fixed: each agent changes its `group` field on every interaction, with equal probability either incrementing `group` or setting it to 1. We show that, no matter the initial group values, after $O(\log n)$ time the group values will be in the range $[1, 8 \log n]$ WHP. Furthermore, the distribution of `group` values is very close to that of n i.i.d. geometric random variables, in the sense that each agent's `group` value is independent of every other, with expected $n/2^i$ agents having $\texttt{group} = i$ if each agent has had at least i interactions.[3]

The agents store an array of "signal" integers in their `signals` field to track the existing `group` values in the population. Each agent in the i'th group is responsible for *boosting* the signal associated with i. The goal is to have $\texttt{signals}[i] > 0$ for all agents if and only if some agent has $\texttt{group} = i$.

The detection protocol of [3], explained below, provides a technique for agents to know which groups are still present. Once a signal for group k fades out, the agents speculate that there is no agent with $\texttt{group} = k$. Depending on the current value stored as `estimate` in agents' memory and the value k, this might cause re-calculating the population size. The agents are constantly checking for the changes in the `signals`. They re-compute `estimate` once there is a large gap between `estimate` and the first `group` i with $\texttt{signals}[i] = 0$. We call i the *first missing value* (stored in the field `FMV`).

The `signals` array is updated as follows. An agent with $\texttt{group} = k$ sets $\texttt{signals}[k]$ to its maximum possible value $(3k + 1)$; we call this *boosting*. Other groups k are updated between two agents u, v with $u.\texttt{signals}[k] = a$ and $v.\texttt{signals}[k] = b$ via *propagation* transitions that set both agent's $\texttt{signals}[k]$ to $\max(a - 1, b - 1, 0)$. The paper [3] used a nonuniform protocol where each agent *already* has an estimate of $\log n$. They prove that if the state being detected (in our case, a state with $\texttt{group} = k$) is *absent* and the current maximum signal is c, then all agents will have signal 0 within $\Theta(c)$ time. However, if the state being detected is present, then the boosting transitions (occurring every $O(1)$ units of parallel time on average in the worst case that its count is only 1) will keep the signal positive in all agents with high probability. For this to hold, the maximum value set during boosting must be $\Omega(\log n)$; the nonuniform protocol of [3] uses its estimate of $\log n$ for this purpose.

Crucially, our protocol associates smaller maximum signal values to smaller group values (so many are much smaller than $\log n$), to ensure that a signal does not take abnormally long to get to 0 when its associated group value is missing. Otherwise, if we set each signal value to $\Omega(\log n)$ (based on the agent's current estimate of $\log n$) during boosting, then it would take time proportional to `estimate` (which could be much larger than the actual value of $\log n$) to detect the absence of a `group` value. Thus it is critical that we provide a novel analysis of the detection protocol, showing that the signals for smaller group values $k \ll \log n$ remain present with high probability. This requires arguing that the boosting reactions for such smaller values are happening with sufficiently higher frequency, due to the higher count of agents with $\texttt{group} = k$, compensating for the smaller boosting signal values they use.

3.3 Formal description of loosely-stabilizing counting protocol

The DynamicCounting protocol (Protocol 1) divides agents among several groups via the UpdateGroup subprotocol. The agents update their `group` from i to $i + 1$ with probability 1/2 or reset to group 1 with probability 1/2. The number of agents at each group and the

[3] The difference is that a geometric random variable G obeys $\Pr[G = j] = 1/2^j$ for *all* $j \in \mathbb{N}^+$, but after i interactions an agent u can increment $u.\texttt{group}$ by at most i, so $\Pr[u.\texttt{group} = j] = 0$ if $j \gg i$.

total number of groups are both random variables dynamically changing through time. We show that the total number of groups remains close to $\log n$ at all times with high probability.

The agents start with arbitrary (or even adversarial) `group` values but we show that WHP the set of `group` values will converge to $[1, 8 \log n]$ within $O(\log n)$ time. Additionally, each agent stores an array of $O(\log n)$ signal values in their `signals` field. The goal is to maintain positive values in the `signals`$[i]$ if some agent has `group` $= i$. The agents store the index of the first `group` i with `signals`$[i] = 0$ in their `FMV` field. They use `FMV` as an approximation of $\log n$ and constantly compare it with their `estimate` value.

Depending on the `estimate` value stored in agents' memory, the agents maintain three main phases of computation:

NormalPhase: An agent stays in the NormalPhase as long as there is a small gap between `estimate` and `FMV`: $0.25 \cdot$ `estimate` $\leq$ `FMV` ≤ 2.5`estimate`.

WaitingPhase: An agent switches from NormalPhase to WaitingPhase if it sees a large gap between the `FMV` and `estimate`: `FMV` $\notin \{0.25 \cdot$ `estimate`$, \ldots, 2.5 \cdot$ `estimate`$\}$. The purpose of WaitingPhase is to give enough time to the other agents so that by the end of the WaitingPhase for one agent, with high probability every other agent has also noticed the large gap between the `FMV` and `estimate` and entered WaitingPhase.

UpdatingPhase: During the UpdatingPhase, every agent uses a new geometric random variable and propagates the maximum by epidemic. We set WaitingPhase long enough so that with high probability when the first agent switches to the UpdatingPhase, the rest of the population are all in WaitingPhase. By the end of UpdatingPhase, every agent switches back to NormalPhase.

Below we explain each subprotocol in more detail.

Algorithm 1 DynamicCounting(u, v).

```
for agent ∈ {u, v} do
    UpdateGroup(agent)
SignalPropagation(u, v)
for agent ∈ {u, v} do
    UpdateMV(agent)
    SizeChecker(agent)
    if agent.phase ≠ NormalPhase then
        TimerRoutine(agent)
PropagateMaxEst(u, v)
for agent ∈ {u, v} do
    if agent.phase = NormalPhase then
        agent.estimate ← agent.GRV
```

In every interaction, both sender and receiver update their group according to the rules of the UpdateGroup subprotocol. If we look at the distribution of the `group` values after $O(\log n)$ time, there are about $n/2$ agents in group 1, $n/4$ agents in group 2, and $n/2^i$ agents in group i (see Figure 1). Note that the number of agents in each group decreases exponentially. Still, we ensure that agents with larger `group` values use stronger signals to propagate, since there is less support for those groups.

To notify all agents about the set of all `group` values that are generated among the population, we use the detection protocol of [3] that is also used as a synchronization scheme in [12]. The agents store an integer for each `group` value that is generated by the population. The `signals` is an array of length $\Theta(\log n)$ such that a positive value in index i represents

Algorithm 2 UpdateGroup(u).

$$\text{u.group} \leftarrow \begin{cases} \text{u.group} + 1 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{cases}$$

some agents in the population have generated $\texttt{group} = i$. Note that, as an agent updates its group, it boosts multiple signals based on its group value, e.g., an agent with $\texttt{group} = i$ helped boost all the indices $1, 2, 3, \ldots, i$ of signals in its last i interactions. We use the SignalPropagation protocol to keep the signal of group i positive as long as some agents have generated $\texttt{group} = i$.

Algorithm 3 SignalPropagation(u, v).

▷ Boosting:
$\text{u.signals}[\text{u.group}] \leftarrow (3 \cdot \text{u.group}) + 1$
$\text{v.signals}[\text{v.group}] \leftarrow (3 \cdot \text{v.group}) + 1$
▷ Propagate signal:
for $i \in \{1, 2, \ldots, \texttt{Max}(|\text{u.signals}|, |\text{v.signals}|\}$ **do**
  $m \leftarrow \texttt{Max}(\text{u.signals}[i], \text{v.signals}[i])$
  $\text{u.signals}[i], \text{v.signals}[i] \leftarrow \texttt{Max}(0, m - 1)$

Regardless of the initial configuration, the distribution of group values changes immediately (in $O(\log n)$ time), but it might take more time for the signals to get updated. It takes $O(i)$ time for $\texttt{signals}[i]$ to hit zero. The larger the index i, $\texttt{signals}[i]$ leaves the population slower. Hence, the agents look at the *first missing signal* that they observe among the array of all signals.

Algorithm 4 UpdateMV(u).

▷ Find the first appearance of a zero in $u.\texttt{signals}$ beyond index $\lceil \texttt{log}(\text{u.estimate}) \rceil$
$s \leftarrow \lceil \texttt{log}(\text{u.estimate}) \rceil$
$\text{u.FMV} \leftarrow \min\{i \in [s, |\text{u.signals}|] \mid \text{u.signals}[i] = 0\}$

Once there is a large gap between the first missing group (FMV) and the agents' estimation of $\log n$ (estimate), each agent individually moves to a *waiting phase* and waits for other agents to catch the same gap between their estimate and FMV. Note that we time this phase as a function of FMV and not the estimate since the estimate is not valid anymore and might be much smaller or larger than the actual value of $\log n$.

Eventually, all agents will notice the large discrepancy between FMV and estimate and move to the WaitingPhase. The WaitingPhase is followed by the UpdatingPhase (explained in the TimerRoutine). In the UpdatingPhase, all agents generate one geometric random variable (stored in GRV) and propagate the maximum value. We assume the agents generate a geometric random variable in one line (line 4 in *Protocol* 6) for simplicity.[4]

Once the UpdatingPhase is completed, all agents will update their estimate to the maximum geometric random variable they have seen and switch to the NormalPhase again. Recall that the agents remain in the NormalPhase as long as their FMV and estimate are relatively close. They continue changing their group values and send group signals as described earlier.

[4] Alternatively, the agents could generate a geometric random variable through $O(\log n)$ consecutive interactions, each selecting a random coin flip (**H** or **T**). In this alternative version, we should make the WaitingPhase longer.

Algorithm 5 SizeChecker(u).

if u.phase = NormalPhase and u.FMV $\notin \{0.25 \cdot$ u.estimate$, \ldots, 2.5 \cdot$ u.estimate$\}$ **then**
  u.phase $\leftarrow$ WaitingPhase ▷ Waiting for other agents to detect the size change

Algorithm 6 TimerRoutine(u).

u.timer $\leftarrow$ u.timer $+ 1$
if u.timer $> 12 \cdot$ u.FMV **then**
  if u.phase = WaitingPhase **then**
    u.GRV $\leftarrow$a new geometric random variable
    u.timer $\leftarrow 0$, u.phase $\leftarrow$ UpdatingPhase
  if u.phase = UpdatingPhase **then**
    u.estimate $\leftarrow$ u.GRV
    u.timer $\leftarrow 0$, u.phase $\leftarrow$ NormalPhase

Intuitively, for each group value, about $n/2^i$ agents will hold group $= i$, and boost signals$[i]$ by setting it to the max $= \Theta(i)$. As the value of i grows, the number of agents with group $= i$ decreases, but their signals get stronger since the agents enhance a group signal i proportional to i. In a normal run of the protocol, the agents expect to have positive values in signals$[i]$ for group values between $[\log \ln n, \log n]$.

4 Analysis of Dynamic Counting Protocol

4.1 Bound on the group values

Recall that the agents calculate a dynamic group value by following the rules of Protocol 2. As described in this protocol, the agents either move to the next group or return to group 1 with probability $1/2$.[5]

In this part, we analyze the distribution of group values. Note that the group values are rather chaotic at the beginning of the protocol since the agents might start holding any arbitrary group values that are much larger than $\log n$. However, after all agents reset back to group $= 1$, we can show for each group $= k$, $\Pr[\text{group} = k] \approx \frac{1}{2^k}$.

In the rest of this section, we assume the initialized setting for simplicity. Later, we show how we can generalize our results to any arbitrary initial configuration. We define $G_{u,t}$ as the group value of agent u at time t and $I(t,u)$ to represent the number of interactions involving this agent by the time t. Note that with this definition, $G_{u,t}$ is equal to k (for $k < I(t,u)$) if and only if agent u generates the sequence of $[HTTT \ldots T]$ (**H** followed by $k-1$ **T**s) during its last k interactions. Thus, we have:

$$\forall k \in \mathbb{N}, \quad 1 \leq k < I(t,u) : \Pr[G_{u,t} = k] = \frac{1}{2^k} \tag{1}$$

With this definition $G_{u,t}$ is undefined for any agents that has not generated H yet. In other words, the values $G_{u,t}$ are "close to geometric" in the sense that they are independent and have probability equal to a geometric random variable on all values $k < I(t,u)$.

[5] The truncated version of this Markov chain (mapping all states $k+1$, $k+2$, ... to $k+1$) is also known as the "winning streak" [30].

Algorithm 7 PropagateMaxEst(u, v).

```
if u.phase = v.phase & u.phase ≠ WaitingPhase then
    u.GRV, v.GRV ← max(u.GRV, v.GRV)
```

▶ **Observation 4.1.** *For agents $u_1, u_2, \ldots, u_n$, and the values $k_i < I(t, u_i)$, for $1 \le i \le n$:*

$$\Pr\left[G_{u_1,t} = k_1, G_{u_2,t} = k_2, \ldots, G_{u_n,t} = k_n\right] = \prod_{i=1}^{n} \Pr\left[G_{u_i,t} = k_i\right]$$

Next we bound the maximum `group` value that has been generated by any agent. Let $M_t = \max_{u \in \mathcal{A}} G_{u,t}$ be the maximum value of $G_{u,t}$ across the population at time t. A proof of the following lemma appears in the full version [20].

▶ **Lemma 4.2.** *Let $c \ge 2$ and let t be a time such that all agents have at least $c \log n$ interactions. In a population of size n, $\frac{1}{d} \log n \le M_t$ with probability at least $1 - \exp\left(-n^{1-1/d}\right)$ and $M_t < c \log n$ with probability at least $1 - n^{1-c}$.*

Note that the maximum `group` value has a large variance. However, we can prove a tight bound for the first `group` value with no support; since to have $\texttt{FMV} = k$, for all values i that are less than k, $\exists u \in \mathcal{A}$ such that $u.\texttt{group} = i$.

So, we analyze the bounds for the first `group` value with no support, i.e., the value $\min\{k \in \mathbb{N}^+ \mid (\forall u \in \mathcal{A})\ u.\texttt{group} \neq k\}$. Considering n i.i.d. geometric random variables, the *first missing value* to be the smallest integer not appearing among the random variables. The first missing value has been studied in the literature [28, 29, 32] as the "the first empty urn" (see also "probabilistic counting" [23]) but for simplicity we use a loose bound for our analysis. The proof appears in the full version [20].

▶ **Lemma 4.3.** *Let $\delta > 0$, $0 < \epsilon < 1$ and let t be a time such that all agents have at least $(1+\delta) \log n$ interactions. Define $\texttt{FMV}_t = \min\{k \in \mathbb{N} \mid (\forall u \in \mathcal{A})\ u.\texttt{group} \neq k\}$ at time t. Then, $\texttt{FMV}_t > (1-\epsilon) \log n$ with probability at least $(1-\epsilon) \log(n) \cdot \exp\left(-n^{\epsilon}\right)$ and $\texttt{FMV}_t \le (1+\delta) \log n$ with probability at least $1 - \left(\frac{1}{n^{\delta/2}}\right)^{(2+\delta)\log n}$.*

4.2 Distribution of the groups

So far, we have proved bounds on the existing `group` values. However, in general, we need to show that at a given time $t = \Omega(\log n)$, there are about $\frac{n}{2^k}$ agents having $\texttt{group} = k$ WHP. The following lemma gives us a lower and upper bound for the number of agents in each `group`:

▶ **Lemma 4.4.** *Let $c \ge 2$, $0 < \epsilon < 1$, $0 \le \delta \le 1$, and let t be a time such that all agents have at least $c \log n$ interactions. Let $1 \le k \le (1-\epsilon) \log n$, then, the number of agents who hold $\texttt{group} = k$, is at least $L_k = (1-\delta)\frac{n}{2^k}$ with probability at least $1 - \exp\left(-\frac{\delta^2 \cdot n^{\epsilon}}{2}\right)$ and at most $U_k = (1+\delta)\frac{n}{2^k}$ with probability at least $1 - \exp\left(-\frac{\delta^2 \cdot n^{\epsilon}}{3}\right)$.*

Proof sketch. The fraction of agents with $\texttt{group} = k$ is equal to the fraction of heads in of a binomial distribution $\mathsf{B}\left(n, 2^{-k}\right)$ with $\mu = \frac{n}{2^k}$, so the Chernoff bound applies. A complete proof is given in the full version [20]. ◀

The following theorem summarizes what we will use later about the distribution of the `group` values and the number of agents residing in each `group` at time t.

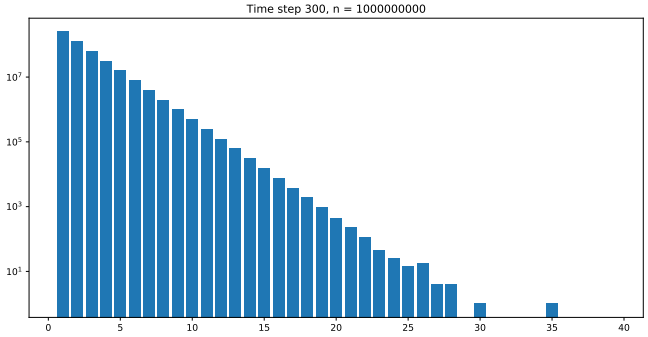

Figure 1 Showing the distribution of `group` values after 300 parallel-time in a population of size $n = 10^9$. The x-axis indicates the different `group` values while the y-axis indicates the number of agents in each group. Note that, we are using log-scale for the y-axis. In this snapshot of the population, $\texttt{FMV} = 29$. Even though, the maximum `group` value is 35 and is much larger than `FMV`.

▶ **Theorem 4.5.** *Fix a time $t \geq d \ln n$ for $d > 30$, let M_t^* and $\texttt{FMV}_t$ be the maximum* `group` *value and the* `FMV` *at this time respectively. Then,*

- $0.9 \log n \leq M_t^* < 0.1d \log n$ *with probability at least* $1 - 2 \cdot n^{1-d/10} - 2 \cdot n^{1-\frac{2d}{3}}$.
- $0.9 \log n \leq \texttt{FMV}_t < 3 \log n$ *with probability at least* $1 - 4 \cdot n^{1-\frac{2d}{3}}$.
- *The number of agents who hold* $\texttt{group} = k$ *for* $1 \leq k \leq 0.9 \log n$, *is in* $\left[\frac{3 \cdot n}{2^{k+2}}, \frac{5 \cdot n}{2^{k+2}}\right]$ *with probability at least* $1 - 4 \cdot n^{1-\frac{2d}{3}}$.

4.3 Group detection

In the previous section, we show that the set of present `group` values among the population will quickly (in $O(\log n)$ time) enter a small interval of values ($[1, 8 \cdot \log n]$) consistent with the population size. In this section, we will prove the following:

- The agents *agree* about the presence of group values in $[\log \ln n, 0.9 \log n]$ after $O(\log n)$ time WHP.
- For a non-existing group value i, each agent will have $\texttt{signals}[i] = 0$ in $O(i + \log n)$ time WHP.

We designed Protocol 3 such that each agent in the i'th `group` *boosts* the associated signal value by setting $\texttt{signals}[i] = B_i$ (recall $B_i = \Theta(i)$). We will show by having at least L_i agents boosting $\texttt{signals}[i]$, the whole population learns about the existence of the i'th `group` in $O(\log n)$ time with high probability. Intuitively, although $\texttt{signals}[i]$ starts lower than $\texttt{signals}[j]$ for $i < j$, so potentially dies out more quickly, it is also boosted more often since more agents have group value i. Concretely, with L_i agents responsible to boost signal i, and for all indices $\log \ln n < i < 0.9 \log n$ in the `signals` of the agents, $\Pr[\texttt{signals}[i] = 0] < \exp\left(-\frac{2^{B_i}}{n/L_i}\right)$.

Intuitively, the next lemma shows that if the `group` values are distributed as in Lemma 4.4, then the whole population will learn about all the present `group` values above $\log \ln n$ within $O(\log n)$ time. Note that $\Pr[u.\texttt{signals}[i] = j]$ is the probability that the agent u has value j in the i^{th} index of its `signals`. The following lemma is a restatement from [3, Section 5.1]. The proof appears in the full version [20].

▶ **Lemma 4.6.** *In the execution of Protocol 3, suppose that for each group value* $\log \ln n < i < 0.9 \log n$, *at least A_i agents hold* $\texttt{group} = i$. *For every agent $u \in \mathcal{A}$ let* $u.\texttt{signals}[i] = r_i$ *when* $u.\texttt{group} = i$. *Assuming each agent has at least r_i interactions, then for a fixed agent u and index i,* $\Pr[u.\texttt{signals}[i] = 0] \leq \left(1 - \frac{A_i}{n}\right)^{2^{r_i - 1}}$.

To use the previous lemma, we need to make sure that the agents wait for sufficiently long time such that each agent has at least r_i interactions. The next corollary uses Lemma 4.6 to derive bounds for the entire protocol using bounds from Lemma 4.4 for the distribution of the `group` values. Also, Corollary 4.7 takes a union bound over *all* agents and group values i, and uses the concrete value $r_i = B_i = 3 \cdot i + 1$ used in our protocol. The proof appears in the full version [20].

▶ **Corollary 4.7.** *For all $i > 0$ and for every agent $u \in \mathcal{A}$, assuming $B_i = 3 \cdot i + 1$ let $u.\texttt{signals}[i] = B_i$ if $u.\texttt{group} = i$. Suppose that for each group value $\log \ln n < i < 0.9 \log n$, at least L_i agents hold $\texttt{group} = i$. Let $\beta \geq 8$; then after $\beta \log n$ time, we have:*

$$\Pr\left[(\exists u \in \mathcal{A})(\exists i \in \{\log \ln n, \ldots, 0.9 \log n\})\ u.\texttt{signals}[i] = 0\right] \leq 2 \cdot n^{1-0.9\beta}$$

Finally, we show that when there is no agent holding $\texttt{group} = i$, then $\texttt{signals}[i]$ will become zero in all agents "quickly" with an arbitrarily large probability. To be precise, with no agent boosting signal i, $\Pr[u.\texttt{signals}[i] = 0] \geq 1 - n^{-\alpha}$ within $\Theta(B_i + \alpha \ln n)$ time WHP in which B_i is the maximum value for signal i. The lemma is a restatement from [17, Lemma 3.3] and [3, Lemma 1].

▶ **Lemma 4.8.** *For every agent $u \in \mathcal{A}$ let $u.\texttt{signals}[i] = B_i$ when $u.\texttt{group} = i$. Assume that no agent sets its `group` to i from this point on. Then for all $\alpha \geq 1$, all agents will have $\texttt{signals}[i] = 0$ after $3n \ln\left(n^{\alpha} \cdot 3^{B_i}\right)$ interactions with probability at least $1 - n^{-\alpha}$.*

Proof. Set $t = 3n \ln\left(n^{\alpha} \cdot 3^{B_i}\right)$ and $R_{max} = B_i$ in the proof of [17, Lemma 3.3]. ◀

4.4 Dynamic size counting protocol analysis

Recall that `estimate` denote the estimate of $\log n$ in agents' memory, and n is the true population size. In the previous section, we show that the set of present `group` values among the population will quickly (in $O(\log n)$ time) enter a small sub-interval of consecutive values in $[1, 3 \cdot \log n]$ consistent with the population size. This section will show that the `group` values will remain in that interval (with high probability for polynomial time). Moreover, the following two lemmas show how the agents update their `estimate` if it is far from $\log n$. The proofs appear in the full version [20].

Assuming the agents' `estimate` is much smaller than $\log n$, the next lemma shows that all the agents will notice the large gap between `estimate` and `FMV`. Hence, they will re-calculate their population size estimate.

▶ **Lemma 4.9.** *Let $M = \max_{u \in \mathcal{A}} u.\texttt{estimate}$. Assuming $M \leq 0.22 \log n$, then the whole population will enter* WaitingPhase *in $O(\log n)$ time with probability at least $1 - O(n^{-2})$.*

For the other direction, assume the population size estimate in agents' memory is much larger than $\log n$. We prove in the following lemma that all the agents will notice the large gap between `estimate` and `FMV`. Hence, they will re-calculate their population size estimation.

Note that in Corollary 4.7, we proved for all `group` values i for $\log \ln n \geq i$, the $\texttt{signals}[i]$ will have a positive value in $O(\log n)$ time. However, we could not prove the same bound for values less than $\log \ln n$. So, inevitably the agents ignore their `signals` for values that are less than $\log \ln n$. Since the agents have no access to the value of $\log n$, they have to use `estimate` as an approximation of $\log n$. Thus, they ignore indices that are less than $\log M$ in `signals`: making `FMV` a function of $\max(\log M, \log n)$. For example, if the true population size is n but $M > 2^n$, then the agents should ignore the appearance of a zero in their `signals` for all indices i that are $\leq \log(M) = n$. The correct `FMV` happens at index

$j = \Theta(\log n)$, but the agents stay in the NormalPhase as long as $\texttt{signals}[i]$ for $i \geq n$ are positive. In this scenario, it takes $O(n)$ time for the agents to switch to WaitingPhase since for each $\texttt{signals}[i]$, it takes $O(i)$ time to hit zero.

This scenario is inevitable with our current detection scheme since for indices i that are less than $\log \ln n$, the event of $\texttt{signals}[i] = 0$ happens frequently.

▶ **Lemma 4.10.** *Let $M = \max_{u \in \mathcal{A}} u.\texttt{estimate}$. Assuming $M \geq 7.5 \cdot \log n$, then the whole population will enter* WaitingPhase *in $O(\log n + \log M)$ time with probability at least $1 - O(n^{-2})$.*

In the next theorem (full proof given in [20]), we will show once there is a large gap between the maximum `estimate` among the population and the true value of $\log n$, the agents update their estimate in $O(\log n + \log M)$ time.

▶ **Theorem 4.11.** *Let $M = \max_{u \in \mathcal{A}} u.\texttt{estimate}$. Assuming $\texttt{estimate} \geq 7.5 \log n$ or $\texttt{estimate} \leq 0.2 \log n$, then every agent replaces its `estimate` with a new value that is in $[\log n - \log \ln n, 2 \log n]$ with probability $1 - O(1/n)$ in $O(\log n + \log M)$ time.*

Proof sketch. By Lemmas 4.9 and 4.10, once an agent notices the large gap between `estimate` and `FMV`, they switch to WaitingPhase. We set WaitingPhase long enough so when the first agent moves to UpdatingPhase, there is no agent left in the NormalPhase. Thus, they all re-generate a new geometric random variable and store the maximum as their `estimate`. ◀

In the full version of the paper [20], we show that the holding time of our protocol is polynomial. For the state complexity, recall that the adversary can initialize agents with large integer values to arbitrarily increase memory usage. Therefore, we should calculate the agents' memory concerning the fields defined in our protocol and the value that the adversary can set in them. Thus, we use s as the largest integer value that the adversary set in the agents' memory.

▶ **Theorem 4.12.** *Let $M = \max u.\texttt{estimate}$ for all $u \in \mathcal{A}$. There is a uniform leaderless loosely-stabilizing population protocol that WHP:*

1. *If $M > 7.5 \log n$ or $M < 0.2 \log n$ reaches to a configuration with all agents set their `estimate` with a value in $[\log n - \log \ln n, 2 \log n]$ in $O(\log n + \log M)$ parallel time.*
2. *If $0.75 \log n < M < 2.25 \log n$, then the agents hold a stable `estimate` during the following $O(n^{15})$ parallel time.*
3. *Assuming for every agent $u \in \mathcal{A}$, $\max(u.\texttt{estimate}, u.\texttt{GRV}, u.\texttt{group}, u.\texttt{signals}.size()) < s$ in the initial configuration, then the protocol uses $O(\log^2(s) + \log n \log \log n)$ bits per agent.*

4.5 Space optimization

In this section, we explain how to reduce the space complexity of the protocol from $O(\log^2(s) + \log n \log \log n)$ to $O(\log^2(s) + (\log \log n)^2)$ bits per agent.

In Protocol 1, the agents keep track of all the present `group` values using an array of size $O(\log n)$ (stored in `signals`) by mapping every $\texttt{group} = i$ to $\texttt{signals}[i]$. We can reduce the space complexity of the protocol by reducing the `signals`' size. Let the agents map a $\texttt{group} = i$ to $\texttt{signals}[\lfloor \log i \rfloor]$. So, instead of monitoring all $O(\log n)$ group values, they keep $O(\log \log n)$ indices in their `signals`. Thus, reducing the space complexity to $O(\log^2(s) + (\log \log n)^2)$ bits per agent.

Recall that in Protocol 1, there are $\approx \frac{n}{2^i}$ agents with $\texttt{group} = i$ for $i \leq 0.9 \log n$ that help keep $\texttt{signals}[i]$ positive. However, with this technique, there will be $\approx \sum_{i=2^j}^{2^{j+1}} \frac{n}{2^i}$ agents that are helping $\texttt{signals}[i]$ to stay positive. So, every lemma in Section 4.3 about Protocol 3 holds. Finally, we update Protocol 5 so that the agents compare their $\texttt{estimate}$ with $2^{\texttt{LFMV}}$ in which $\texttt{LFMV}$ is the smallest index $i > \log \log M$ such that $\texttt{signals}[i] = 0$. On the negative side of this optimization, we get a less sensitive protocol with respect to the gap between agents' $\texttt{estimate}$ and $\log n$.

▶ **Theorem 4.13.** *Let $M = \max u.\texttt{estimate}$ for all $u \in \mathcal{A}$. There is a uniform leaderless loosely-stabilizing population protocol that WHP:*

1. *If $M > 15 \log n$ or $M < 0.1 \log n$ reaches to a configuration with all agents set their $\texttt{estimate}$ with a value in $[\log n - \log \ln n, 2 \log n]$ in $O(\log n + \log M)$ parallel time.*
2. *If $0.75 \log n < M < 2.17 \log n$, then the agents hold a stable $\texttt{estimate}$ during the following $O(n^{15})$ parallel time.*
3. *Assuming for every agent $u \in \mathcal{A}$, $\max(u.\texttt{estimate}, u.\texttt{GRV}, u.\texttt{group}, u.\texttt{signals}.size()) < s$ in the initial configuration, then the protocol uses $O(\log^2(s) + (\log \log n)^2)$ bits per agent.*

5 Conclusion and open problems

In this paper, we introduced the dynamic size counting problem. Assuming an adversary who can add or remove agents, the agents must update their $\texttt{estimate}$ according to the changes in the population size. There are several open questions related to this problem.

Reducing convergence time. Our protocol's convergence time depends on both the previous (n_{prev}) and next (n_{next}) population sizes, though exponentially less on the former: $O(\log n_{\text{next}} + \log \log n_{\text{prev}})$. Is there a protocol with optimal convergence time $O(\log n_{\text{next}})$?

Increasing holding time. Observation 3.4 states that the holding time must be finite, but it is likely that much longer holding times than $\Omega(n^c)$ for constant c are achievable. For the loosely-stabilizing leader election problem, there is a provable tradeoff in the sense that the holding time is at most exponential in the convergence time [26, 36]. Does a similar tradeoff hold for the dynamic size counting problem?

Reducing space. Our main protocol uses $O(s + (\log n)^{\log n})$ states (equivalent to $O(\log^2(s) + \log n \log \log n)$ bits). In Section 4.5, we showed how we can reduce the state complexity of our protocol to $o(n^\epsilon)$ (equivalent to $O(\log^2(s) + (\log \log n)^2)$ bits) by mapping more than one $\texttt{group}$ to each index of the $\texttt{signals}$. With this trick, we reduce the size of the $\texttt{signals}$ from $O(\log n)$ to $O(\log \log n)$. Another interesting idea is to replace our $O(\log n)$ detection scheme to $O(1)$ detection protocol of [22] which puts a constant threshold on the values stored in each index. So, it may be possible to reduce the space complexity even more to $O(c^{O(\log n)})$ (with all $O(\log n)$ indices present) or $O(c^{O(\log \log n)}) = \text{polylog}(n)$ (using our optimization technique to have $O(\log \log n)$ indices in the $\texttt{signals}$).

However, the current protocol of [22] has a one-sided error that makes it hard to compose with our protocol. With probability $\epsilon > 0$, the agents might say signal i has disappeared even though there exists agents with $\texttt{group} = i$ in the population.

Additionally, in the presence of a uniform self-stabilizing synchronization scheme, one could think of consecutive rounds of independent size computation. The agents update their output if the new computed population size drastically differs from the previously computed

population size. Note that the self-stabilizing clock must be independent of the population size since we allow the adversary to change the value of $\log n$ by adding or removing agents. To the best of our knowledge, there is no such synchronization scheme available to population protocols.

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, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *DNA Computing and Molecular Programming*, pages 155–171. Springer International Publishing, 2017.

4 Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *ICALP 2015: Proceedings, Part II, of the 42nd International Colloquium on Automata, Languages, and Programming - Volume 9135*, pages 479–491. Springer-Verlag, 2015. doi:10.1007/978-3-662-47666-6_38.

5 Dan Alistarh, Martin Töpfer, and Przemysław Uznański. Fast and robust comparison in population protocols. In *PODC 2021: The ACM Symposium on Principles of Distributed Computing*, 2021.

6 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, 2006.

7 Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. *ACM Trans. Auton. Adapt. Syst.*, 3(4):1–28, 2008. doi:10.1145/1452001.1452003.

8 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.

9 Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In *Distributed Computing*, pages 631–646, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

10 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, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

11 Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. An $O(\log^{3/2} n)$ parallel time population protocol for majority with $O(\log n)$ states. In *PODC 2020: Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 191–199. Association for Computing Machinery, 2020. doi:10.1145/3382734.3405747.

12 Petra Berenbrink, Felix Biermeier, Christopher Hahn, and Dominik Kaaser. Loosely-stabilizing phase clocks and the adaptive majority problem. In *SAND 2021: 1st Symposium on Algorithmic Foundations of Dynamic Networks*, 2021.

13 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A Population Protocol for Exact Majority with $O(\log^{5/3} n)$ Stabilization Time and Theta(log n) States. In *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.DISC.2018.10.

14 Petra Berenbrink, George Giakkoupis, and Peter 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, pages 119–129, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384312.

15 Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and efficient leader election. In *SOSA 2018: The 1st Symposium on Simplicity in Algorithms*, pages 9:1–9:11, 2018. doi:10.4230/OASIcs.SOSA.2018.9.

16 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. Association for Computing Machinery, 2017. doi:10.1145/3087801.3087858.

17 Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In *PODC 2021: Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 33–44. ACM, 2021. doi:10.1145/3465084.3467898.

18 David Doty and Mahsa Eftekhari. Efficient size estimation and impossibility of termination in uniform dense population protocols. In *PODC 2019: Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 34–42. Association for Computing Machinery, 2019. doi:10.1145/3293611.3331627.

19 David Doty and Mahsa Eftekhari. A survey of size counting in population protocols. *Theoretical Computer Science*, 894:91–102, 2021. Building Bridges – Honoring Nataša Jonoska on the Occasion of Her 60th Birthday. doi:10.1016/j.tcs.2021.08.038.

20 David Doty and Mahsa Eftekhari. Dynamic size counting in population protocols, 2022. arXiv:2202.12864.

21 David Doty, Mahsa Eftekhari, Othon Michail, Paul G. Spirakis, and Michail Theofilatos. Brief Announcement: Exact Size Counting in Uniform Population Protocols in Nearly Logarithmic Time. In *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:3, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.DISC.2018.46.

22 Bartłomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 87–99, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188818.

23 Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

24 Leszek Gąsieniec, Grzegorz Stachowiak, and Przemysław Uznański. Almost logarithmic-time space optimal leader election in population protocols. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, pages 93–102. Association for Computing Machinery, 2019. doi:10.1145/3323165.3323178.

25 Shafi Goldwasser, Rafail Ostrovsky, Alessandra Scafuro, and Adam Sealfon. Population stability: regulating size in the presence of an adversary. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 397–406. ACM, 2018.

26 Taisuke Izumi. On space and time complexity of loosely-stabilizing leader election. In *Structural Information and Communication Complexity*, pages 299–312. Springer International Publishing, 2015.

27 Tomoko Izumi, Keigo Kinpara, Taisuke Izumi, and Koichi Wada. Space-efficient self-stabilizing counting population protocols on mobile sensor networks. *Theoretical Computer Science*, 552:99–108, 2014. doi:10.1016/j.tcs.2014.07.028.

28 Guy Louchard and Helmut Prodinger. The moments problem of extreme-value related distribution functions. *Algorithmica*, 2004.

29 Guy Louchard, Helmut Prodinger, and Mark Daniel Ward. The number of distinct values of some multiplicity in sequences of geometrically distributed random variables. In *Discrete Mathematics and Theoretical Computer Science*, pages 231–256. Discrete Mathematics and Theoretical Computer Science, 2005.

30 László Lovász and Peter Winkler. Reversal of markov chains and the forget time. *Combinatorics, Probability and Computing*, 7(2):189–204, 1998.

31 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.

32 Helmut Prodinger. Philippe flajolet's early work in combinatorics. *arXiv preprint*, 2021. arXiv:2103.15791.

33 Yuichi Sudo, Ryota Eguchi, Taisuke Izumi, and Toshimitsu Masuzawa. Time-optimal loosely-stabilizing leader election in population protocols. In *DISC 2021: The 35th International Symposium on Distributed Computing*, 2021.

34 Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Loosely-stabilizing leader election in population protocol model. In *Structural Information and Communication Complexity*, pages 295–308, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

35 Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Logarithmic expected-time leader election in population protocol model. In *Stabilization, Safety, and Security of Distributed Systems*, pages 323–337. Springer International Publishing, 2019.

36 Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, Toshimitsu Masuzawa, Ajoy K. Datta, and Lawrence L. Larmore. Loosely-Stabilizing Leader Election with Polylogarithmic Convergence Time. In *22nd International Conference on Principles of Distributed Systems (OPODIS 2018)*, volume 125 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.OPODIS.2018.30.