# Message Complexity of Population Protocols

**Talley Amir**
Yale University, New Haven, CT, USA
https://cpsc.yale.edu/people/talley-amir
talley.amir@yale.edu

**James Aspnes**
Yale University, New Haven, CT, USA
https://www.cs.yale.edu/homes/aspnes/
james.aspnes@gmail.com

**David Doty**
University of California, Davis, CA, USA
https://web.cs.ucdavis.edu/~doty/
doty@ucdavis.edu

**Mahsa Eftekhari**
University of California, Davis, CA, USA
https://eftekhari.cs.ucdavis.edu/
mhseftekhari@ucdavis.edu

**Eric Severson**
University of California, Davis, CA, USA
https://www.math.ucdavis.edu/~severson/
eseverson@ucdavis.edu

## Abstract

The standard population protocol model assumes that when two agents interact, each observes the entire state of the other. We initiate the study of **message complexity** for population protocols, where an agent's state is divided into an externally-visible message and externally-hidden local state.

We consider the case of $O(1)$ message complexity. When time is unrestricted, we obtain an exact characterization of the stably computable predicates based on the number of internal states $s(n)$: If $s(n) = o(n)$ then the protocol computes semilinear predicates (unlike the original model, which can compute non-semilinear predicates with $s(n) = O(\log n)$), and otherwise it computes a predicate decidable by a nondeterministic $O(n \log s(n))$-space-bounded Turing machine. We then introduce novel $O(\text{polylog}(n))$ expected time protocols for junta/leader election and general purpose broadcast correct with high probability, and approximate and exact population size counting correct with probability 1. Finally, we show that the main constraint on the power of bounded-message-size protocols is the size of the internal states: with unbounded internal states, any computable function can be computed with probability 1 in the limit by a protocol that uses only *1-bit* messages.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms

**Keywords and phrases** population protocol, message complexity, space-optimal

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2020.6

**Related Version** https://arxiv.org/abs/2003.09532

**Supplementary Material** https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/eftekhari-mhs/population-protocols

**Funding** The second author was supported by NSF award CCF-1650596. The third, fourth, and fifth authors were supported by NSF awards 1619343, 1844976, 1900931.

## 1 Introduction

Population protocols, introduced by Angluin, Aspnes, Diamadi, Fischer, and Peralta [6], are a class of algorithms that model ad hoc networks of finite-state mobile agents. At each step, a pair of agents is picked uniformly at random to interact, each observing the other's state and updating its own state in response. The original model [6] limited agents to $O(1)$ states, independent of the population size $n$. This limited the computational power (to only semilinear predicates [7]) and the time efficiency in performing fundamental tasks (e.g. the linear-time lower bound for leader election [28]). Recent work uses $\omega(1)$ states, yielding more time-efficient algorithms for fundamental tasks (e.g. [1–4, 15, 30–32, 41]). Is the improved performance a result of higher communication throughput or greater storage capacity?

Whereas the original model supposes that agents can view the entirety of the other's local state upon interacting, we introduce a new variant of this model which distinguishes a segment of the agent's state that is externally visible to its interacting partner, called the **message**. This variant generalizes previous work in the context of consensus that examines the particular case of **binary signaling** [5, 36], where the message is limited to a single bit. We study the computational power of population protocols that have $O(1)$ message complexity and varying local state complexity, ranging from $O(1)$ to unbounded.

### 1.1 Motivation

The population protocol framework was conceived to model passively mobile ad hoc sensor networks. In this setting the amount of communication bandwidth can be a tighter constraint than the local computation performed by a sensor. These two constraints – bandwidth efficiency and energy efficiency – are viewed as distinct in the networking literature. In some scenarios it makes more sense to optimize for one or the other, or to strike a balance [23,33,42]. The restriction to $O(1)$ messages but $\omega(1)$ internal states is germane when the communication in an interaction is more costly than the accompanying local computation.

Synthetic chemistry is another domain in which population protocols are an appropriate abstract model of computation. This is a subclass of chemical reaction networks, which are known to have similar computational power [21, 38]. Using a physical primitive known as **DNA strand displacement** [44], *every* chemical reaction network with $O(1)$ species (**states** in the language of population protocols) can be theoretically implemented by a set of DNA complexes [39], justifying the use of chemical reactions as an implementable programming language. This approach has been used to synthesize nontrivial chemical systems in the wet lab, resulting in pure DNA implementations of a chemical oscillator [40] and the "approximate majority" population protocol [9, 22]. Some theoretical [37] and experimental [43] systems are able to assemble unbounded-length heteropolymers, best modeled using arbitrarily many states (exponential in the polymer length) but only $O(1)$ messages rendering the smaller "locally visible region" near one or both ends of the polymer.

Finally, our model of $\omega(1)$ internal states and $O(1)$ external messages is a natural mathematical intermediate between the original $O(1)$-state model and the more recent $\omega(1)$-state model. Population protocols with superconstant states are provably more powerful [20], so it is intrinsically interesting to determine how powerful this new intermediate model is.

## 1.2 Our Contribution

**Table 1** Summary of positive results: Above, the event of “not error” means that the answer is correct *and* the stated time and state bounds hold, unless the error probability is 0, in which case it refers only to the output being correct. In that case, the time and state bounds are in expectation, but still hold with high probability: in all cases, the probability of error is $O(1/n)$. (It should be straightforward to extend to $O(1/n^k)$ for any $k$, but for simplicity in proof statements we fix $k = 1$. We rely on [14, Theorem 1] that, as stated, holds only for a fixed exponent $k$, though it seems a more detailed analysis could achieve arbitrary $k$.) Note that when the probability of computing the correct output is 1 (i.e. the protocol stabilizes), the Time column denotes time to convergence. State complexities are accurate with high probability. $|M|$ is the number of messages, either a constant larger than 1, or exactly 2 (1-bit). Compute $\log n$ means computing either $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$. In the first row $t_P$ is the expected convergence time for $P$.

| Problem solved | Pr[error] | Time | States | $\|M\|$ | Leader |
|---|---|---|---|---|---|
| Simulate $s(n)$-state open protocol (Corollary 3.4) | 0 | $O(t_P n^2 \log s(n))$ | $O(s(n)^2)$ | $O(1)$ | Yes |
| Junta election (Theorem 4.1) | $> 0$ | $O(\log^2 n)$ | $O(\log^2 n)$ | 1-bit | No |
| Compute $n$ (Theorem 4.2) | $> 0$ | $O(\log^2 n)$ | $O(n \log^2 n)$ | $O(1)$ | Yes |
| Compute $\log n$ (Corollary 4.3) | $> 0$ | $O(\log^2 n)$ | $O(\log n)$ | $O(1)$ | Yes |
| Stably compute $n$ (Corollary 4.4) | 0 | $O(\log^2 n)$ | $O(n^4 \log^4 n)$ | $O(1)$ | Yes |
| Leaderlessly compute $n$ (Corollary 4.5) | $> 0$ | $O(\log^2 n)$ | $O(n \text{ polylog}(n))$ | $O(1)$ | No |
| Leaderlessly compute $\log n$ (Corollary 4.5) | $> 0$ | $O(\log^2 n)$ | $O(\text{polylog}(n))$ | $O(1)$ | No |
| Compute $d$-input predicate (Corollary 4.6) | $> 0$ | $O(d \log^2 n)$ | $O(n^d \log^2 n)$ | $O(1)$ | Yes |
| TM simulation (Theorem 5.2) | 0 | unbounded | unbounded | 1-bit | No |

We introduce this new variant of population protocols and show three main results:

We first (Section 3) completely resolve the question of the computational power of $O(1)$ messages, with Theorem 3.2. In the positive direction, with poly$(n)$ states ($O(\log n)$ bits), we give a simulation of $\Omega(1)$-bit messages (Theorems 3.3 and 3.5). Corollary 3.9 is an asymptotically sharp negative result: $O(1)$-message, $o(n)$-state ($\log n - \omega(1)$ bits) protocols compute only semilinear predicates.

Secondly (Section 4), we focus on time-efficient computation. We develop novel $O(\log^2 n)$-time algorithms for junta election (the key primitive to leader election) and exact population size counting (naturally suited to this model, where $O(n)$ local states and $O(1)$ messages are the minimal power to make this problem solvable). The counting protocol can specialize with fewer states to estimate the size (count $\log n$), and also generalize with more states to count the entire input configuration (so any predicate can be locally computed).

Thirdly (Section 5), we explore the extreme limits of the model where message complexity is limited to 1 bit. We construct a 1-bit broadcast primitive, showing it is powerful enough to simulate a Turing Machine with probability 1 correctness using unbounded local memory.

## 1.3 Comparison to existing work and new techniques required

Most protocols using $\omega(1)$ states [1–3, 12, 13, 16–19, 25, 30, 31, 34, 35, 41] crucially use $\omega(1)$-size messages. Key transitions in such protocols involve comparing two integers/ids of size $\omega(1)$ in a single step, which is not possible with $O(1)$-size messages. Sending a superconstant-size message over multiple interactions is not efficient (though it is a trick we employ for unbounded time results such as Theorem 3.3), since there is not enough time for the two agents to wait for another interaction (which takes $\Theta(n)$ expected time), nor is there any way to distinguish each other in future interactions. We introduce new techniques that rely on timing of internal counters to get around this limitation.

Our *JuntaElection* protocol (see paper's extended version) is our primary fast leaderless protocol, used to make other leader-driven protocols leaderless. It elects a **junta**, a group of $O(\sqrt{n})$ agents, in $O(\log^2 n)$ time. As with many other existing protocols [15,30,31,41], this is used for a junta-driven **phase clock** [8] allowing agents to synchronize in a downstream computation. The cited protocols have agents choose an integer "level" $\ell$, propagating by epidemic the maximum level ($\Theta(\log\log n)$ [15,30,31] as in our case, or $\Theta(\log n)$ [41]). Agents who chose the maximum level are in the junta. Lacking the ability to communicate the levels in 1-bit messages, we rely on **timing** of agents' internal counters to detect whether a higher level exists: Agents with level $\ell$ count up to $\approx 4^\ell$, (roughly) telling all other agents to continue counting and stop at $\approx 4^\ell$, unless another agent (with high probability with a higher level) tells them to continue counting. The actual details require intricate choice of timing and analysis to conclude that all agents stop at the same counter value with high probability. We push the technique of communication via timing further, showing that *1-bit* messages suffice to elect a leader, broadcast arbitrary messages, and simulate a Turing Machine.

The **majority** problem is that of deciding which of two opinions in a population is more numerous. One population protocol [5] distinguishes between external messages and internal state, using 1-bit messages (**binary signaling**) to achieve consensus in expected $O(nr \log nr)$ interactions; however, this protocol uses $O(r)$ internal states, where $r$ is a tunable parameter independent of $n$ and can thus be considered constant, making this a $O(1)$-state solution to the majority problem. Other existing protocols [2, 4, 12, 13, 18] use an $O(1)$-message "doubling/cancelling" technique, which works on top of a synchronization primitive, but these protocols use $\omega(1)$ messages to achieve synchronization. Our $O(1)$-message junta-election protocol can be composed with the doubling/cancelling technique to give a high-probability, $O(1)$-message majority protocol, which, unlike [2,4,13,18], is **uniform**, requiring no estimate of $n$.

Indeed, all of our protocols are uniform in this sense, in contrast to several existing $\omega(1)$-state protocols [1–4,13,16,18,19,31,34,35,41]. Many of our protocols could be simplified greatly by allowing nonuniformity. Briefly, an estimate of $\log n$ within a constant factor would allow agents to synchronize themselves using a **leaderless phase clock** based on counting to $c\log n$ for some large constant $c$. the lack of such synchronization is a major challenge in devising correct, efficient $O(1)$-message protocols.

## 2 Model

We write $\log n$ to denote $\log_2 n$, and $\ln n$ to denote $\log_e n$. The original population protocol model [6] involves a population of $n$ **agents**, each of which holds a **state** in a **state space** $Q$. Interactions between agents update the states of both agents according to a **transition function** $\delta : Q \times Q \to Q \times Q$, where interactions are asymmetric: in each interaction, one of the agents is the **initiator** of the interaction, and one the **responder**.

We consider a refinement of the model in which the state of an agent is explicitly divided into an internal component that is not visible to other agents, and an external component that is. The internal component of the state is drawn from the state space $I$ and the external component, or **message**, is drawn from a message space $M$. The set of states $Q$ is the

Cartesian product $I \times M$. The transition function $\delta$ is modified to enforce the restriction that an agent in an interaction cannot observe the internal state of the other agent: $\delta$ is now a function from $Q \times M \times \{\mathsf{initiator}, \mathsf{responder}\}$ to $Q$. When an agent in state $q_1 = \langle i_1, m_1 \rangle$ initiates an interaction with an agent in state $q_2 = \langle i_2, m_2 \rangle$, the new states of the agents are given by $q_1' = \langle i_1', m_1' \rangle = \delta(q_1, m_2, \mathsf{initiator})$ and $q_2' = \langle i_2', m_2' \rangle = \delta(q_2, m_1, \mathsf{responder})$.

The set of producible states $Q(n)$ and the set of producible messages $M(n)$ can both depend on $n$. The function $s : \mathbb{N} \to \mathbb{N}$ defined as $s(n) = |Q(n)|$ is the **state complexity** of a population protocol. The function $n \mapsto |M(n)|$ is the **message complexity**. If $|I| = 1$ and each agent's state is merely defined by its message (the original model [6] and its superconstant state generalization), we say the protocol is **open**, so $|Q(n)| = |M(n)|$ for all $n$. We will mostly be interested in population protocols with modest state complexity (at most polynomial in $n$, and often only polylogarithmic in $n$) and constant message complexity. Given two functions $s, m : \mathbb{N} \to \mathbb{N}$, a **$s(n)$-state, $m(n)$-message population protocol** is one with state complexity $s$ and message complexity $m$. Note that the complexity bounds we discuss are *worst-case*: $s(n)$ is the most number of states that can be produced in any population of size $n$ under any execution. We place high probability bounds on the state complexity (e.g. *JuntaElection* where agents generate a geometric random variable which may take on any positive integer value). These do not consider the set of producible states, so our impossibility results (Theorem 3.8) on state and message complexity do not apply.

*Problems solved by population protocols.* A **configuration** gives the state of all agents. Population protocols have some problem-dependent notion of "correct" configurations. For example, for **leader election** a configuration with a single leader is correct. For computation of a **predicate** $\phi : \mathbb{N}^d \to \{\mathsf{yes}, \mathsf{no}\}$ (a.k.a., **decision problem**), the initial state of each agent is from a $d$-element subset $\Sigma$ of states, states are partitioned into two subsets representing "yes" and "no", and a configuration is correct if all agents give the answer $\phi(\vec{i})$, where $\vec{i} \in \mathbb{N}^d$ represents the initial counts of agents in each state in $\Sigma$. A population protocol is **leader-driven** if its states have a Boolean field $\mathsf{leader} \in \{L, F\}$ (i.e. the state set $Q = \{L, F\} \times Q'$), such that in every valid initial configuration, exactly one agent has $\mathsf{leader} = L$.

*Time complexity.* For measuring time complexity, we assume **random scheduling**, where at each interaction two agents are chosen uniformly at random from all $n(n-1)$ possible ordered pairs of agents. Time complexity is defined by **parallel time**, the number of interactions divided by $n/2$ which we henceforth simply refer to as **time**. This definition reflects the average number of interactions in which an agent participates, and reflects an assumption that agents effectively interact in parallel, even though for simplicity of analysis this parallelism is modeled by interleaving interactions sequentially.

*Convergence/stabilization.* A configuration $\vec{c}$ is **stably correct** if every configuration reachable from $\vec{c}$ is correct. An execution $\mathcal{E} = (\vec{c}_0, \vec{c}_1, \ldots)$ is picked at random according to the scheduler explained above. We say $\mathcal{E}$ **converges** (respectively, **stabilizes)** at interaction $i \in \mathbb{N}$ if $\vec{c}_{i-1}$ is not correct (resp., stably correct) and for all $j \geq i$, $\vec{c}_j$ is correct (resp., stably correct). The **(parallel) convergence/stabilization time** of a protocol is the number of interactions to converge/stabilize, divided by $n/2$. Convergence can happen strictly before stabilization, although a protocol with finite reachability (i.e. for each $\vec{c}$, finitely many configurations are reachable from $\vec{c}$) converges from $\vec{c}$ with probability $p \in [0, 1]$ if and only if it stabilizes from $\vec{c}$ with probability $p$. For a computational task $T$ equipped with some definition of "correct", we say that a protocol **stably computes $T$ with probability** $p$ if, with probability $p$, it stabilizes (equivalently, converges).[1]

[1] Let $C$ and $S$ respectively be the set of stabilizing and converging executions. Then $\Pr[C \setminus S] = 0$. Suppose a protocol converges in an execution $(\vec{c_0}, \vec{c_1}, ...)$ at interaction $i$. If it did not stabilize, then for

## 3 Computability with unrestricted time

In this section we study $s(n)$-state, $O(1)$-message protocols where time is not restricted. Theorem 3.2 is our main result in this section, which completely characterizes the power of such protocols in terms of the number of bits required to store the states.

Let $\mathsf{CMPP}(f(n))$ be the set of all predicates stably computed by an $s(n)$-state, $O(1)$-message population protocol, where $s(n) = 2^{O(f(n))}$ (using $O(f(n))$ bits of memory). Let $\mathsf{SNSPACE}(g(n))$ be the set of all predicates $\phi : \mathbb{N}^d \to \{0,1\}$ decidable by a nondeterministic $O(g(n))$-space-bounded Turing machine, when inputs are given in unary.[2] The results of [20] considered a similar complexity class $\mathsf{PMSPACE}(f(n))$ of stably computable predicates using $O(f(n))$ bits of memory and $O(f(n))$ bit messages.[3] Let $\mathsf{SL}$ be the set of all **semilinear** predicates [7]. Their main result is the following characterization:

▶ **Theorem 3.1** ( [20]). *Let $f : \mathbb{N} \to \mathbb{N}$. If $f(n) = o(\log\log n)$, then $\mathsf{PMSPACE}(f(n)) = \mathsf{SL}$. If $f(n) = \Omega(\log n)$, then $\mathsf{PMSPACE}(f(n)) = \mathsf{SNSPACE}(n \cdot f(n))$.*

Since the memory is expressed in Theorem 3.1 as number of *bits* (exponentially smaller than number of **states**), the multiplicative constants hidden in the Big-O notation become polynomial-factor terms in number of states. Theorem 3.2 is a similar dichotomy theorem for $O(1)$-message population protocols, which is sharper in that it holds for *all* values of $f(n)$. The proof is in the full paper version, and follows from Theorems 3.3, 3.5, 3.8, and 3.1.

▶ **Theorem 3.2.** *Let $f : \mathbb{N} \to \mathbb{N}$. If $f(n) = o(\log n)$, then $\mathsf{CMPP}(f(n)) = \mathsf{SL}$, otherwise $\mathsf{CMPP}(f(n)) = \mathsf{SNSPACE}(n \cdot f(n))$.*

### 3.1 Leader-driven $O(s(n)^2)$-state, $O(1)$-message protocols can simulate open $s(n)$-state protocols

In this section we show that $O(s(n)^2)$-state, $O(1)$-message, leader-driven protocols can simulate $s(n)$-state open protocols (whether leader-driven or not). Thus, allowing a leader and ignoring quadratic differences in state complexity, there is no difference whatsoever between the computational power of $O(1)$-message protocols and open protocols. Theorem 3.3 proves the general case of $m(n)$-message protocols. Corollary 3.4 is the special case of **open** protocols, where $s(n) = m(n)$. The simulation incurs a time slowdown of factor $n^2 \log m(n)$, where $n$ is the population size and $m(n) \leq s(n)$ is the message complexity of the simulated protocol, so it ports (non-sublinear) computability results from the open protocol model.

Intuitively, the construction of Theorem 3.3 chooses two agents to "mark" as $\mathsf{initiator}$ and $\mathsf{responder}$, which then successively pass a bit string as they interact, until they have transmitted the full message of size $\log m(n)$ bits. Crucially, starting with a leader allows only one simulated transition to be taking place at a time.

all $j > i$, some incorrect configuration $\vec{d_j}$ would be reachable from $\vec{c_j}$. Let $p_j > 0$ denote the probability of reaching $\vec{d_j}$ from $\vec{c_j}$. The set of reachable configurations is bounded with probability 1, so $\min_{j>i} p_j$ is well-defined and positive. The probability of never reaching any $\vec{d_j}$ is then 0.

[2] In [20] these are called **symmetric** predicates on the assumption that the $d$ counts in $\vec{i} \in \mathbb{N}^d$ are presented to the Turing machine as a $\|\vec{i}\|$-length string of symbols from an input alphabet $\Sigma$ with $|\Sigma| = d$, with the same answer on all permutations of the string.

[3] In fact, to obtain their positive result for large space bounds, they do not need fully open protocols. Their simulation of nondeterministic $nf(n)$-space-bounded Turing machines just requires $O(\log n)$ bit messages to exchange unique IDs, even if $f(n) = \omega(\log n)$.

▶ **Theorem 3.3.** *For every $s(n)$-state, $m(n)$-message protocol $P$, there is a leader-driven, $O(s(n) \cdot m(n))$-state, $O(1)$-message protocol $S$ that simulates $P$, and each interaction of $P$ takes expected $O(n^2 \log m(n))$ interactions of $S$ to simulate.*

The next corollary applies to **open** protocols, where each agent's message is its full state.

▶ **Corollary 3.4.** *For every $s(n)$-state, open population protocol $P$, there is a leader-driven, $O(s(n)^2)$-state, $O(1)$-message population protocol $S$ that simulates $P$, and each interaction of $P$ takes expected $O(n^2 \log s(n))$ interactions of $S$ to simulate.*

It is known that $\Omega(\log n)$-state open protocols have computational power beyond that of $O(1)$-state protocols (limited to semilinear predicates [7] and functions [21]), and Corollary 3.4 grants this same computational power to leader-driven $O(1)$-message protocols. Theorem 3.8 in subsection 3.4 shows that Corollary 3.4 crucially depends on the assumption of an initial leader in the simulating protocol, by demonstrating that **leaderless** $O(1)$-message, $o(n)$-state protocols are no more powerful than $O(1)$-state open protocols.

## 3.2 Leader election can be composed with leader-driven, $s(n)$-state, $O(1)$-message protocols using $O(n^3 \log n)$ state overhead

Leader election is possible in linear time with 1-bit messages by "fratricide": $\ell, \ell \to \ell, f$. A downstream leader-driven protocol $P$ will not work unaltered if composed with this leader election, because the presence of multiple leaders prior to convergence causes incorrect transitions of $P$. A straightforward fix using $O(n)$ messages involves exact size counting via transitions $\ell_i, \ell_j \to \ell_{i+j}, f_{i+j}$ (requiring $\Omega(n)$ messages) and each transition between agents with respective values $n$ and $i < n$ resets the latter agent to its initial state in $P$. As soon as the last agent is reset with value $n$, the protocol faithfully executes a **tail** of an execution of $P$ from $\vec{i}$, i.e. an execution starting at a configuration $\vec{c}$ reachable from $\vec{i}$. Thus if $P$ is correct with probability 1, the composed protocol is also correct with probability 1. Theorem 3.5 demonstrates a similar "composition by resetting" strategy using $O(1)$ messages.

▶ **Theorem 3.5.** *For any leader-driven, $s(n)$-state, $O(1)$-message protocol $P$, there is a leaderless, $O(s(n)n^3 \log n)$-state, $O(1)$-message protocol $S$ that, after $O(n \log n)$ expected time, executes a tail of an execution of $P$.*

Theorem 3.5 depends crucially on using $\geq n$ states, since Theorem 3.8 shows leaderless, $O(1)$-message, $o(n)$-state protocols are no more powerful than $O(1)$-state open protocols.

## 3.3 Deterministic Broadcast

The construction used in our composable leader election protocol (see extended paper) can be modified to also give the leader the ability to stably broadcast a message to the entire population. After the last restart, the leader agent $a$ counts the entire population by moving them between phases. We can view these phases as deterministically synchronized rounds (each expected time $O(n \log n)$ [10]). Add a field $\mathsf{bit} \in \{0, 1\}$ to the message. The leader $a$ can then communicate a bit string to the population by sending one bit during each round. This lets the population stably compute the population size $n$, by having the leader send $n$ as a bit string in $O(\log n)$ rounds (stabilizing in expected $O(n \log^2 n)$ time). It uses $O(\log n)$ state overhead to store the bits it has broadcast, so $O(n^3 \log^2 n)$ states total. Thus we conclude:

▶ **Corollary 3.6.** *There is an $O(n^3 \log^2 n)$-state, $O(1)$-message protocol that stably computes the population size $n$ (storing in every agents state), in expected $O(n \log^2 n)$ time.*

## 3.4 Leaderless $o(n)$-state, $O(1)$-message protocols compute only semilinear predicates

Theorem 3.8 is broad and does not apply to a particular "mode of computation" (e.g., deciding predicates [6, 7], computing functions [11, 21, 27], leader election [15, 29]). It does, however, assume a problem-specific notion of **valid** initial configurations.[4] We say a protocol is **additive** if the set of valid initial configurations is closed under addition. This rules out, for instance, protocols with an initial leader. Indeed, Corollary 3.9 is false if an initial leader is allowed, by applying Theorem 3.3 to let a leader-driven $O(1)$-message protocol simulate any $o(n)$-state open protocol that stably computes a non-semilinear predicate/function.[5]

A lower bound result in [20] shows that with an absolute space bound of $o(\log n)$ states, their model is limited to only stably computing the semilinear predicates.[6] The core of their argument bounds the number of reachable memory states.

▶ **Theorem 3.7** ( [20]). *Let $s : \mathbb{N} \to \mathbb{N}$ and consider an additive, $s(n)$-state, open population protocol. Then either $s(n) = O(1)$ or $s(n) = \Omega(\log n)$.*

As a corollary, if $s(n) = o(\log n)$, then $s(n)$ is in fact constant, reducing to the original $O(1)$-state model, which can only stably compute semilinear predicates [7]. We use a similar proof technique to show an exponentially stronger result in the model of $O(1)$ messages.

▶ **Theorem 3.8.** *Let $s : \mathbb{N} \to \mathbb{N}$ and consider an additive, $s(n)$-state, $O(1)$-message population protocol. Then either $s(n) = O(1)$ or $s(n) = \Omega(n)$.*

**Proof sketch.** A fixed population $\vec{i}_c$ suffices to produce any of the $O(1)$ messages. Consider a population $\vec{i}_n$ of size $n$. If $s(n) \neq O(1)$, then for some state $b$ not producible from $\vec{i}_n$, $b$ **is** producible by sending some message $m$ to a state $a$ producible from $\vec{i}_n$ (though $a$ and $m$ cannot appear *simultaneously* in a configuration reachable from $\vec{i}_n$). By combining $\vec{i}_n$ with $\vec{i}_c$, we have a population of size $n + O(1)$ that can produce $b$. Thus the number of producible states grows at least linearly with $n$. ◀

Population protocols using $O(1)$ states compute only semilinear predicates [7], resulting in the following corollary. Since we require additivity of valid initial configurations, the corollary applies only to leaderless protocols.

▶ **Corollary 3.9.** *If a leaderless, $o(n)$-state, $O(1)$-message protocol stably computes a predicate $\phi$, then $\phi$ is semilinear.*

## 4 Computability with polylogarithmic time complexity

In this section we study $O(1)$-message protocols with "fast" computation (polylog$(n)$ time).

[4] For example, for leader election, all agents have the same initial state. For computation of predicates [6] or functions [11, 21], all agents represent "input" from a constant alphabet, with possibly an extra leader.

[5] For example, transitions $(i; \ell), (i; \ell) \to (i + 1; \ell), (i + 1; f)$ and $(j; \ell), (i; f) \to (j; \ell), (j; f)$, which starting from all agents in state $(1, \ell)$, give each agent the value $\lfloor \log n \rfloor$.

[6] Theorem 14 of [20] states "$o(\log \log n)$" bits, which implies $o(\log n)$ states, though the converse does not hold. However, inspecting their proof reveals that the result holds up to $\log(n) - 1$ states.

## 4.1 High-probability junta election using 1-bit messages

In this section, we describe a uniform protocol using 1-bit messages that, with high probability, elects a "junta" of $O(\sqrt{n})$ agents in polylogarithmic time. The protocol also lets each agent compute an integer $k \in \mathbb{N}^+$ that is the same for all agents and is one of $\lfloor \log\log n \rfloor$, $\lceil \log\log n \rceil$, or $\lceil \log\log n \rceil + 1$. Thus $2^k$ is an estimate of $\log n$ within a multiplicative factor 2.

Furthermore, *JuntaElection* is composable, in that we can use the protocol as a black box to initialize other protocols that require either a junta for a phase clock, or an approximation of $\log n$ (e.g. for a leaderless phase clock). Thus any nonuniform protocol that requires $k$-bit messages can be composed with *JuntaElection* to achieve a uniform protocol using $(k+1)$-bit messages with an additive time overhead of $O(\log^2 n)$. For example, we can compose *JuntaElection* with the the leader election protocol of [30] using $\frac{1}{2}$-coin flips to convert the $O(\sqrt{n})$-size junta to size 1, i.e. elect a leader, in expected $O(\log^2 n)$ time and $O(1)$ messages, or with majority protocols that use $O(1)$ messages for doubling/cancelling phases, synchronized by the junta-driven phase clock [12]. Our protocol has a positive probability of failure. It is yet unknown whether there is an $O(1)$-message protocol that stably approximates $\log n$ or elects a junta of size $n^\epsilon$ for some $\epsilon \in (0,1)$ in sublinear stabilization time.

**High-level description of protocol.** The protocol is described formally in the extended paper. Intuitively, it works as follows. Most leader/junta election protocols generate an **id**, where the agents generating the maximum id are the junta. In our protocol, we generate an id called **level**, but $O(1)$ messages prevent direct communication of levels, so we employ a timing-based strategy to communicate the maximum level using only messages `Go` and `Stop`.

Each agent initially generates a local geometric random variable $G$ (number of fair coin flips until the first heads, i.e., an immediate heads results in $G = 1$) and computes its **level** as $\lceil \log G \rceil$. (We can also use synthetic coin techniques [1] to simulate fair coin flips and increment their level from $i$ to $i+1$ as they flip $2^i$ consecutive tails.)

We define consecutive disjoint intervals $G_0, R_0, G_1, R_1, \ldots \subset \mathbb{N}$ (**green** and **red**) partitioning the natural number line. We call $R_i$'s last element $d_i = \max R_i$ a **door**. (See Figure 1, formal definition below.) Each agent keeps a local counter, initially 0, that is incremented on some interactions. An agent is **in round** $i$ if its counter is in $G_i \cup R_i$. The goal is to get every agent to count up until the round equal to the maximum level $k$ generated by any agent and stop its counter at $d_k$. An agent with level $l$ in round $i$ is **eager** if $i < l$ and **cautious** otherwise. Intuitively, eager agents race through doors until their own level, telling all other agents to keep going, but become cautious at and beyond their own level, advancing past a door into the next round only if another agent tells them to do so (via a message $m = $ `Go`). More formally, an eager agent always sends a message of `Go` and increments its counter on every interaction. A cautious agent sends message `Go` if and only if its counter is in $G_i$ for some $i$, increments its counter on every interaction in $G_i \cup R_i \setminus \{d_i\}$ unconditionally, and increments its counter beyond $d_i$ if and only if the other agent's message is `Go`. Agents drop out of the junta when they leave their own level, so (assuming no agent leaves the maximum level) those who generated the maximum level are the eventual junta.

To formally define the intervals, let $c \in \mathbb{N}^+$. Each $G_i$, with $|G_i| = c4^i$, is called a **green** interval, $R_i$, with $|R_i| = \frac{3c}{2}4^i$, a **red** interval. Note that $d_i = \sum_{j=0}^{i-1}(|G_j| + |R_j|) = c\left(1 + \frac{3}{2}\right)\frac{4^i - 1}{4-1} < \frac{5c}{6}4^i = \frac{5}{6}|G_i|$, so $|G_i|$ is larger than the union of all the previous intervals by a constant multiplicative factor. The max level $k$ is $\Theta(\log\log n)$ with high probability, so its corresponding interval $|G_k| = \Theta(4^{\log\log n}) = \Theta(\log^2 n)$. Thus, with high probability, the agents use $O(\log^2 n)$ states for their counters and stop at the door $d_k$ after $O(\log^2 n)$ time.

To compose *JuntaElection* with a downstream protocol $P$, agents can simply restart $P$ whenever they move beyond a $d_i$, and then wait to start simulating $P$ until they reach the next $d_{i+1}$ (Restarting is a common technique in distributed computing for composition and is not original to this paper, e.g., [30].) In the early stages of *JuntaElection*, the downstream protocol gets restarted many times, but eventually, all agents will move past $d_{k-1}$, after which they will restart the downstream protocol for the last time. The agents will all simultaneously be in the last interval $G_k \cup R_k$ before stopping at $d_k$. Thus all simulated downstream interactions of $P$ will be between agents that agree on $k$.

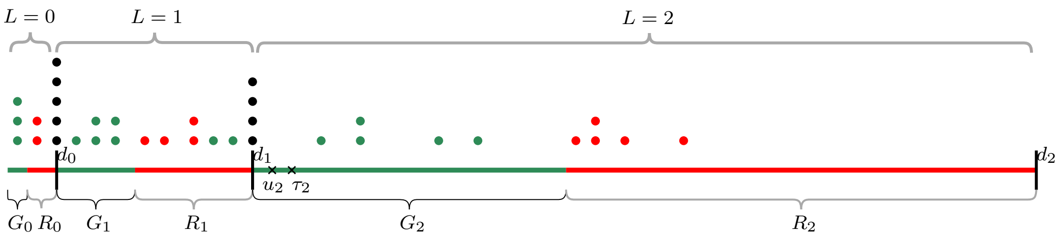

**Figure 1** Agents, represented as dots, increment their counters through the $G_0, R_0, G_1, R_1, G_2, R_2$ intervals. Agents in green intervals or any interval before their own level have message `Go`. Agents in red intervals at their own level or later have message `Stop`. At the end of a red interval (the door $d_i$, shown with black horizontal line) at their own level or later, the agents (black dots) wait to increment their counter until they see a message `Go`. The special times marked $u_i, \tau_i$ are used in proving correctness.

▶ **Theorem 4.1.** *With probability $1 - O(1/n)$, JuntaElectionuses $O(\log^2 n)$ states and elects a junta of size $O(\sqrt{n})$ in $O(\log^2 n)$ time, after which $v$.*`count` $= d_k$ *for all agents $v$, where $k \in \{\lfloor \log \log n \rfloor, \lceil \log \log n \rceil, \lceil \log \log n \rceil + 1\}$.*

**Proof sketch.** We must show the agents are synchronized. When the interval lengths are $\Omega(\log n)$, we argue that the agents' local counters are bounded within the same interval. The main challenge is reasoning about agents that may be stuck at a door.

Our argument shows that a constant fraction $n/4$ of agents stay synchronized in each green interval, up until near the max level. Then, we argue that during the later green intervals, straggler agents are able to catch up, because they have a constant probability of passing through each door and the length of the green interval is more than the sum of all previous intervals. We then show the entire population in synchronized within the last few intervals. Thus all agents will have a `Stop` message when the population reaches the final door $d_k$, and the agents will stop their counters at $d_k$. See extended paper for full proof. ◀

Our proof techniques require setting $|G_i| = 700 \cdot 4^i$. However, simulation results show successful convergence when $|G_i| = 16 \cdot 2^i$ (see extended paper). Scaling the intervals this way lets $|G_M| = \Theta(\log n)$, so the protocol takes $O(\log n)$ time and $O(\log n)$ internal states.

## 4.2 Leader-driven, $O(\log^2 n)$-convergence-time exact size counting

In this section we give a $O(\log^2 n)$ time, high-probability protocol for a problem that is natural for agents with superconstant memory: exact population size counting. The probability of error can be reduced to 0 with standard techniques; see Corollary 4.4. This problem has been studied in the context of open protocols in both the exact [17, 26] and approximate [17, 25] settings, where it is known that open protocols can approximate $n$ within multiplicative factor 2, by computing either $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$, using $O(\log n \log \log n)$ states and $O(\log^2 n)$

time [17]. Open protocols can compute the **exact** value of $n$, using $O(n \log n \log\log n)$ states and $O(\log n)$ time [17]. Both protocols can be made probability-1, with a $O(\log n)$ factor increase in states, i.e. $O(\log^2 n \log\log n)$ states for calculating $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$, and $O(n \log^2 n \log\log n)$ states for exactly computing $n$. However, note that our results below are leader-driven, so direct comparison with the leaderless results of [17] is not appropriate.

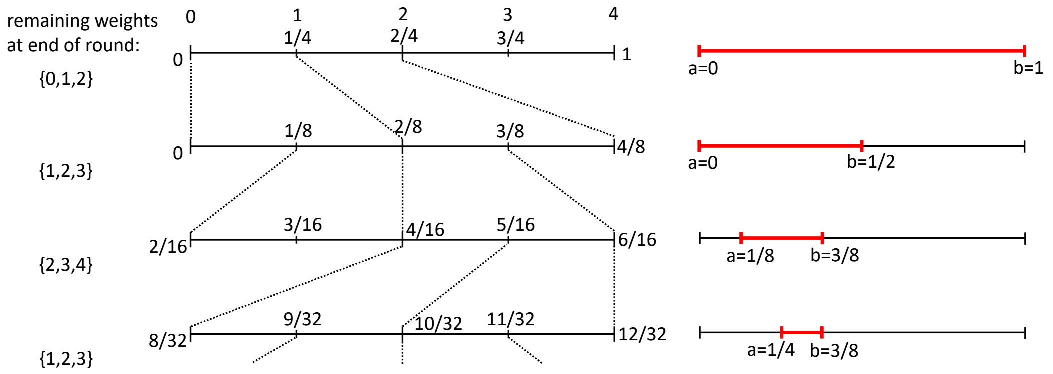

**Figure 2** Update rule for fast exact counting protocol. All agents start with a mass of 0 and weight $w = 0$, except the leader, who starts with mass $= 1$ and $w = 4$. They conduct averaging on weight $w$ for one round, at which point (with high probability) three consecutive weights remain. The figure shows how the remaining masses map to the next subinterval, with the weight $w$ updating to $2(w - w_{\min})$ where $w_{\min}$ is the minimum value of $w$ at the end of the *Averaging* phase. The right side shows the subintervals to scale. Each agent updates its internal state to represent the interval $[a, b]$. Once $[a, b]$ contains only a single number of the form $\frac{1}{n}$, the protocol terminates, and each agent knows the value $n$. The first $\log n$ rounds would always have 0 as the minimum remaining weights, but we allow other values to show concretely how the updating rule works.

▶ **Theorem 4.2.** *There is an $O(1)$-message leader-driven population protocol that, with probability $1 - O(1/n)$, exactly counts the population size $n$ (storing it in each agent's internal state), in $O(\log^2 n)$ time and using $O(n \log^2 n)$ states.*

**Proof sketch.** The full proof and pseudocode for *ExactCounting* are in the extended paper. The proof uses the "fast averaging" technique employed by other population protocols [4,17, 26,34,35], where each agent holds an integer and computes the transition $i, j \to \left\lfloor \frac{i+j}{2} \right\rfloor, \left\lceil \frac{i+j}{2} \right\rceil$. In the $O(1)$-message setting this will not work exactly as described.

Intuitively, the leader will distribute 1 unit of what we can imagine is a continuous mass into the population. Rational-valued averaging of this mass would result in each agent converging to $1/n$, from which $n$ can be computed, $O(1)$ messages cannot represent arbitrary rationals. Instead, we allow agents to communicate a few bits of their number at a time, while ensuring that before moving on, they agree on an interval containing the true average, which shrinks by half each round, synchronized by a leader-driven phase clock [8]).

Figure 2 shows the updating rule. Each agent's state represents an interval $[a, b] \subseteq [0, 1]$, where $b - a = 2^{-r}$ during round $r \in \mathbb{N}$ (initialized to $r = 0$). $a$ is a dyadic rational, initialized to $a = 0.0$, containing $r + 2$ bits after the binary point. One message field $W = \{0, 1, 2, 3, 4\}$ describes varying amounts of extra weight. The value $w \in W$ counts for $\frac{w}{4 \cdot 2^r}$ units of mass in round $r$. An agent is interpreted as having mass $= a + \frac{w}{4 \cdot 2^r} \in [a, b]$ (note representing $\frac{w}{4 \cdot 2^r}$ is what requires $r + 2$ bits after the binary point). The leader is initialized with $w = 4$ (and mass $= 0 + \frac{4}{4 \cdot 1} = 1$), and the followers are initialized with $w = 0$ (and mass $= 0$).

The full proof shows that, with high probability, every agent will always have the same value of $a$. This implies, via the averaging rule for weights, that mass is conserved and the sum of mass in the population is 1. Thus for all agents at all times, it holds that the true average $\frac{1}{n}$ stays within the interval $[a, b]$. Once the interval contains only a single integer reciprocal $\frac{1}{n}$, the protocol terminates with all agents knowing $n$. ◀

Terminating *ExactCounting* early gives a space efficient protocol for estimating $\log n$:

▶ **Corollary 4.3.** *A leader-driven, $O(1)$-message protocol, with probability $1 - O(1/n)$, computes $r \in \{\lfloor \log n \rfloor, \lceil \log n \rceil\}$, in $O(\log^2 n)$ time using $O(\log n)$ states.*

**Proof sketch.** We run *ExactCounting* until the interval $[a, b]$ contains exactly one power of two $2^{-k}$, and then output $k$, unless it contains no powers of two, in which case we output arbitrarily either of the powers of 2 contained in the interval of the **previous** round. If $n = 2^k$, then $k = \log n$ exactly. Otherwise, since the interval contains no other power of 2, but it contains $1/n$, then $k \in \{\lfloor \log n \rfloor, \lceil \log n \rceil\}$. ◀

By the standard technique of running in parallel with a slower deterministic counting protocol, we can convert our exact counting protocol to have probability 0 of error while retaining fast convergence time (see extended paper for proof).

▶ **Corollary 4.4.** *There are $O(1)$-message, leader-driven population protocols that, with probability 1, respectively count the exact population size n and estimate it by computing $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$, both with expected $O(\log^2 n)$ convergence time and $O(n \log^2 n)$ stabilization time. With probability $1 - O(1/n)$, they use $O(n^4 \log^4 n)$ and $O(\log^2 n)$ states, respectively.*

The next corollary shows that *ExactCounting*can be made leaderless by composing with the leader election protocol derived from *JuntaElection*. Proofs are in the extended paper.

▶ **Corollary 4.5.** *There is a leaderless, $O(1)$-message population protocol that exactly counts the population size n in $O(\log^2 n)$ time and $O(n \operatorname{polylog} n)$ states, succeeding with probability $1-O(1/n)$. There is also a leaderless, $O(1)$-message population protocol that computes $\lfloor \log n \rfloor$ or $\lceil \log n \rceil$ in $O(\log^2 n)$ time and $O(\operatorname{polylog} n)$ states, succeeding with probability $1 - O(1/n)$.*

## 4.3 Leader-driven, $O(\log^2 n)$-time predicate computation

We can use techniques from Theorem 4.2 to show how to compute, using a leader and with high probability, any predicate on a constant alphabet $\Sigma$, up to the space bounds allowed by the agents. We assume that there is one leader agent, and that every other agent has a state from a fixed alphabet $\Sigma$. Exactly the semilinear predicates are computable with probability 1 by $O(1)$-state open protocols [7] (with $> \log n$ states, more predicates are possible [20]).

▶ **Corollary 4.6.** *Let $d \in \mathbb{N}^+$ and let $\Sigma$ be a d-symbol input alphabet. Then there is an $O(1)$ message leader-driven population protocol that, with probability $1 - O(1/n)$, exactly counts the input vector $\vec{i} \in \mathbb{N}^d$ (storing it in each agent's internal state), in $O(d \log^2 n)$ time and using $O(n^d \log^2 n)$ states.*

**Proof sketch.** Agents first run the exact counting protocol of Theorem 4.2 to store locally the value $n$. Agents then use a similar strategy to this protocol to count how many agents have input $x$ for each symbol $x \in \Sigma$. Having now stored the entire initial population's input in their internal state, they can simply compute any computable predicate $\phi$ locally. ◀

If an agent can locally store the entire initial configuration, it can compute any predicate computable by the transition function $\delta$. We required that $\delta$ be computable by a Turing Machine with $O(\log s(n))$ bits of memory, to make our model comparable with [20]. Thus we can compute all predicates computable by $O(\log n)$ bit space-bounded Turing Machines.

## 5 Computability with 1-bit messages

We will show that with 1-bit messages, it is possible to simulate a synchronous system that provides a 1-bit broadcast channel. This will be used to simulate more complex systems. We sacrifice stabilization for convergence and rely on unbounded counters to ensure convergence in the limit with probability 1. Let us begin by defining the simulated system.

A **synchronous broadcast system** consists of $n$ synchronous agents that carry out a sequence of **rounds**. In a broadcast round, each agent generates a 1-bit outgoing message. These messages are combined using the OR function to produce the outcome for this round.

Broadcast operations can be used to detect conditions such as the presence of a leader, or ordinary message transmission if a unique agent is allowed to broadcast in a particular round. However, because broadcast operations are symmetric, they cannot be used for symmetry breaking. For the purpose of electing a leader, we assume that agents have the ability to flip coins; once we have a leader, further agents may be recruited for particular roles using an auxiliary protocol that allows the leader to select a single agent from the population in some round. The broadcast and selection protocols are mutually exclusive: either all agents participate in a broadcast in some round or all agents participate in selection. This is possible by showing that all agents eventually agree on the round number forever with probability 1.

Simulating this model in a population protocol requires (a) enforcing synchrony across agents, so that each agent updates its state consistently with the round structure; (b) implementing the broadcast channel that computes the OR of the agents' outputs; and (c) implementing the selection protocol. We show how to do this in the following section.

### 5.1 Implementing the core primitives

Broadcasts are implemented by epidemics that propagate 1 messages, separated by barrier phases in which all agents display 0. Selection is implemented by having the leader display a 1 to the first agent it meets. Both protocols depend on the number of steps at each agent being approximately synchronized with high probability; after $t(n/2)$ steps, all agents' step counts should be within the range $t \pm O(\sqrt{t \log n})$ with high probability; the time to carry out a broadcast is also $O(\log n)$ with high probability (see proofs in extended paper). By increasing the length of each round over time, the total probability across all rounds of an error occurring in either the broadcast or selection protocol due to out-of-sync agents or slow broadcasts converges to a finite value. Applying the Borel-Cantelli lemma then shows that there is a round after which no further failures occur with probability 1.

#### 5.1.1 Details

Observe that the probability that a particular agent $i$ participates in an interaction is exactly $2/n$, and that the events that $i$ participates in distinct interactions are independent. If we let $X_i^t$ be the indicator variable that agent $i$ participates in the $t$-th interaction, then $S_i^t = \sum_{j=1}^{t} X_i^t$ is a sum of independent Bernoulli random variables, and obeys the Chernoff bound $\Pr\left[|S_i^t - \mu| > \mu\delta\right] < 2e^{-\mu\delta^2/3}$, where $\mu = \mathrm{E}\left[S_i^t\right] = 2t/n$ and $0 \le \delta \le 1$.

The execution of each agent is organized as a sequence of rounds, where each round $r$ for $r = 1, 2, \ldots$ consists of exactly $5r^2$ steps. The first $2r^2$ steps will be a **barrier phase** during which the agent displays message 0 and updates its state during an interaction only by incrementing its step counter. The remaining $3r^2$ steps will be an **interaction phase** in which the agents may execute one of two protocols. In a **broadcast phase**, each agent will propagate an epidemic represented by message 1, recording if it observed such an epidemic and possibly initiating the epidemic itself if instructed to do so by the protocol. In a **selection phase**, a leader agent displays 1 for its first encounter, and the agent interacting with the leader receives a special mark. The choice of broadcast/selection phase is determined by the controlling protocol and is the same for all agents. As in a barrier phase, an agent in an interaction phase continues to update its step counter with each interaction.

The controlling protocol updates the state of the agent at the end of each round. Each agent $v$ has a state $v$.state that is one of broadcasting (agent is initiating a broadcast of value 1) receiving (agent is waiting to detect a 1), received (agent has detected a 1), selecting (agent is attempting to select another agent), candidate (agent is a candidate for selection), selected (agent has been selected), or idle (agent has selected another agent and is now waiting for the end of the round). We assume the controlling protocol assigns consistent values to the agents in each phase: if one or more agents start in state broadcasting, the rest should start in state receiving; while if some agent starts in state selecting, the rest should start in state candidate. Pseudocode for this protocol and its proof of correctness are in the extended paper.

## 5.2 Convergent computation of arbitrary symmetric functions

Early rounds produce incorrect results, so we need an error-recovery mechanism. We describe a basic protocol for electing a leader and having it gather inputs from the other agents. This allows the leader to compute the output of an arbitrary symmetric function and broadcast it to the other agents. The protocol guarantees termination with probability 1 even in executions where some of the rounds exhibit errors in the underlying broadcast mechanism. By restarting the protocol when it terminates, we guarantee that the protocol eventually runs without errors, thus converging to the correct output.

Each agent $v$ has a Boolean field $v$.leader that marks it as a leader (or candidate leader) and a field $v$.processed that marks whether it has reported its input $v$.input to the leader. Agents cycle through 7 rounds, where the round number is $r \bmod 7$, organized as follows:

**Round 0** Any leader broadcasts 1. A non-leader that receives 0 sets its leader bit. This round allows recovery from states with no leaders.

**Round 1** Any leader broadcasts 1 with probability 1/2. A leader that does not broadcast but receives a 1 clears its leader bit.

**Round 2** Any agent that cleared its leader bit in the previous round broadcasts 1. This causes any remaining leaders that receive a 1 to restart the information-gathering protocol and causes any non-leaders that receive a 1 to clear their processed bits. Broadcasting a 1 in this round is also used by the leader to restart the protocol after completion.

**Round 3** Any agent $v$ with $v$.processed $= 1$ broadcasts 1. This is used by the leader and other agents to detect unprocessed inputs.

**Round 4** If a leader received a 1 in the previous round, and there is no transmission in progress from a non-leader agent, the leader executes a selection operation. The selected agent sets its processed bit and transmits its input if its processed bit is not already set. If the processed bit is set, the agent transmits nothing in the following two rounds.

**Rounds 5 and 6** These are used to transmit either (a) one bit of a selected agent's input, or (b) one bit of the protocol output. In either case the bit is encoded as two bits using the convention $01 = 0$, $10 = 1$, $00 =$stop. Note that the absence of a broadcast in both rounds is interpreted as stop, which both allows a selected agent to signal it has already been processed and guarantees eventual termination after an agent finishes transmitting its input even if some of the broadcasts are garbled. Two agents may transmit simultaneously (e.g. if there are multiple surviving leaders), thus agents must be prepared to handle receiving 11. Either we can decide to have agents interpret it as a fixed value: $11 = 1$, or interpret it as a signal to restart the protocol by broadcasting a 1 in the next Round 2.

The protocol terminates when the leader has collected all inputs (detected by the absence of a signal in Round 3) and transmits the computed output to all agents (using Rounds 5 and 6 over however many iterations are needed). We assume that the computed output has finite length for any combination of inputs. After transmitting the output, the leader broadcasts a 1 in Round 2 to restart the information-gathering component of the protocol.

▶ **Lemma 5.1.** *In any execution with finite errors in the underlying broadcast protocol, with probability* 1*, the above protocol converges to a single leader and then restarts infinitely often.*

Lemma 5.1 is proven correct by demonstrating that, as defined, this protocol elects exactly one leader by Round 2 which correctly processes all non-leader agents with probability 1 by the end of Round 6. The full proof can be found in the paper's extended version.

Once the protocol restarts with a single leader, any subsequent error-free execution produces correct output. This follows from the fact that the leader collects the input from every agent exactly once. Since each agent records as its output the last output broadcast by the leader, this causes all agents to converge to holding the correct output with probability 1.

Because the leader has unbounded states, it can simulate an arbitrary Turing machine. This allows the output to converge to the value of any computable symmetric function. The restriction to symmetric functions follows from uniformity of the agents in the initial configuration, but can be overcome, assuming inputs include indexes. We have thus shown:

▶ **Theorem 5.2.** *For any computable symmetric function $f$, there is a population protocol using* 1*-bit messages and unbounded internal states that starts in an initial configuration where each agent $i$ is distinguished only by its input $x_i$, that converges to having each agent holding output $f(x_1, \dots, x_n)$.*

Our construction exploits the unbounded state at each agent to allow the leader to simulate the entire computation. While probability-1 convergence requires unbounded state in the limit (otherwise there is a nonzero probability that any round fails), it may be desirable to put off expanding the state as long as possible. In the extended paper, we argue that with some small tweaks, the construction can be adapted to distribute the contents of a Turing machine tape of $s$ bits across all agents of the population as in [20], reducing the storage overhead at each agent for the Turing machine computation to $O(s/n + \log s)$ bits.

## 6 Open problems

**Probability-1 computation.** Many of our protocols have a positive probability of error. Common techniques for achieving zero error probability in $\omega(1)$-state protocols require $\omega(1)$ messages. Based on this, we conjecture that probability-1 leader election using $O(1)$ messages requires $\Omega(n)$ time to stabilize. This is known to hold for $O(1)$ states [28], though sublinear-time **convergence** is possible with $O(1)$ states [32].

**Time lower bounds.** A tool for time lower bounds (e.g. probability-1 leader election [1,28]) is a "density lemma" [1,24] showing that when the state complexity is $\leq \frac{1}{2} \log \log n$, all states appear in "large" count. This is false for $s(n) > \log \log n$, which is the key to the fastest space-optimal leader election protocols [15,30,31]. A density lemma applies to the **messages** of $O(1)$-message protocols, no matter the state complexity (derivable from [25, Lemma 4.2]). Does this imply that $O(1)$-message leader election requires linear stabilization time?

**Power of 1-bit messages with $O(1)$-states.** $O(1)$-state **open** protocols can stably compute exactly the semilinear predicates [7]. Can all semilinear predicates be stably computed with 1-bit messages? A related question is whether there is a direct simulation of $O(1)$-message protocols by 1-bit message protocols (similar to Theorem 3.3).

**Efficient predicate computation.** Corollary 4.6 can be used to efficiently compute any computable predicate $\phi : \mathbb{N}^d \to \{0, 1\}$ but requires storing the entire initial configuration locally in each agent ($\Theta(n^d)$ states). Corollary 3.6 can be used to compute any computable predicate storing unique IDs in each agent ($O(n)$ states), but it is slow since communication is routed through a leader. What predicates can be computed time- *and* space-efficiently?

## References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *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 *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2221–2239. SIAM, 2018.

3 Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *International Colloquium on Automata, Languages, and Programming*, pages 479–491. Springer, 2015.

4 Dan Alistarh, Rati Gelashvili, and Milan Vojnović. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 47–56, 2015.

5 Dana Angluin, James Aspnes, and Dongqu Chen. A population protocol for binary signaling consensus. Technical Report YALEU/DCS/TR-1527, Yale University Department of Computer Science, 2016.

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

7 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, page 292–299, New York, NY, USA, 2006. Association for Computing Machinery. `doi:10.1145/1146381.1146425`.

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

9 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, July 2008.

10 James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. Time and Space Optimal Counting in Population Protocols. In Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone, editors, *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.OPODIS.2016.13`.

11 Amanda Belleville, David Doty, and David Soloveichik. Hardness of Computing and Approximating Predicates and Functions with Leaderless Population Protocols. In *ICALP 2017:*

*44th International Colloquium on Automata, Languages, and Programming*, volume 80, pages 141:1–141:14, 2017.

12 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Majority & stabilization in population protocols. *arXiv preprint arXiv:1805.04586*, 2018.

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)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

14 Petra Berenbrink, Tom Friedetzky, Dominik Kaaser, and Peter Kling. Tight & simple load balancing. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 718–726. IEEE, 2019.

15 Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *STOC 2020: 52nd Annual ACM Symposium on Theory of Computing*, 2020. to appear.

16 Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and efficient leader election. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

17 Petra Berenbrink, Dominik Kaaser, and Tomasz Radzik. On counting the population size. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 43–52. ACM, 2019. doi:10.1145/3293611.3331631.

18 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 *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 451–453, 2017.

19 Janna Burman, David Doty, Thomas Nowak, Eric E Severson, and Chuan Xu. Efficient self-stabilizing leader election in population protocols. *arXiv preprint arXiv:1907.06068*, 2019.

20 Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. In *Proceedings of the 7th ACM ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing*, FOMC '11, page 6–15, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1998476.1998480.

21 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2014. Special issue of invited papers from DNA 2012. doi:10.1007/s11047-013-9393-6.

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

23 L. H. Diakite and L. Yu. Energy and bandwidth efficient wireless sensor communications for improving the energy efficiency of the air interface for wireless sensor networks. In *2013 IEEE Third International Conference on Information Science and Technology (ICIST)*, pages 1426–1429, March 2013. doi:10.1109/ICIST.2013.6747805.

24 David Doty. Timing in chemical reaction networks. In *SODA 2014: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 772–784. SIAM, 2014.

25 David Doty and Mahsa Eftekhari. Efficient size estimation and impossibility of termination in uniform dense population protocols. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 34–42. ACM, 2019. doi:10.1145/3293611.3331627.

26 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 *DISC 2018: 32nd International Symposium on Distributed Computing*, 2018.

**27** David Doty and Monir Hajiaghayi. Leaderless deterministic chemical reaction networks. *Natural Computing*, 14(2):213–223, 2015. Preliminary version appeared in DNA 2013.

**28** David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018. Special issue of invited papers from DISC 2015.

**29** R. Elsässer and T. Radzik. Recent results in population protocols for exact majority and leader election. *Bulletin of the EATCS*, 126, 2018. URL: `http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546`.

**30** Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *SODA*, pages 2653–2667, 2018. `doi:10.1137/1.9781611975031.169`.

**31** Leszek Gasieniec, Grzegorz Stachowiak, and Przemyslaw Uznanski. Almost logarithmic-time space optimal leader election in population protocols. In *SPAA*, pages 93–102, 2019. `doi:10.1145/3323165.3323178`.

**32** Adrian Kosowski and Przemysław Uznański. Brief announcement: Population protocols are fast. In *PODC*, pages 475–477, 2018. URL: `https://dl.acm.org/citation.cfm?id=3212788`.

**33** R. Lu, X. Lin, H. Zhu, X. Liang, and X. Shen. Becan: A bandwidth-efficient cooperative authentication scheme for filtering injected false data in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(1):32–43, January 2012. `doi:10.1109/TPDS.2011.95`.

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

**35** 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.

**36** E. Perron, D. Vasudevan, and M. Vojnovic. Using three states for binary consensus on complete graphs. In *IEEE INFOCOM 2009*, pages 2527–2535, April 2009. `doi:10.1109/INFCOM.2009.5062181`.

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

**38** D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7:615–633, 2008.

**39** David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. In *DNA14*, pages 57–69, 2008. `doi:10.1007/978-3-642-03076-5_6`.

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

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

**42** P. Udayakumar, R. Vyas, and O. P. Vyas. Energy efficient election protocol for wireless sensor networks. In *2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT)*, pages 1028–1033, March 2013. `doi:10.1109/ICCPCT.2013.6529026`.

**43** Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567:366–372, 2019. `doi:10.1038/s41586-019-1014-9`.

**44** Bernard Yurke, Andrew J. Turberfield, Allen P. Mills, Jr., Friedrich C. Simmel, and Jennifer L. Nuemann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.