**Distributed Computing manuscript No.**
(will be inserted by the editor)

**Dana Angluin · James Aspnes · David Eisenstat**

# Fast Computation by Population Protocols With a Leader

Received: date / Accepted: date

**Abstract** Fast algorithms are presented for performing computations in a probabilistic population model. This is a variant of the standard population protocol model—in which finite-state agents interact in pairs under the control of an adversary scheduler—where all pairs are equally likely to be chosen for each interaction. It is shown that when a unique leader agent is provided in the initial population, the population can simulate a virtual register machine with high probability in which standard arithmetic operations like comparison, addition, subtraction, and multiplication and division by constants can be simulated in $O(n \log^5 n)$ interactions using a simple register representation or in $O(n \log^2 n)$ interactions using a more sophisticated representation that requires an extra $O(n \log^{O(1)} n)$-interaction initialization step. The central method is the extensive use of epidemics to propagate information from and to the leader, combined with an epidemic-based phase clock used to detect when these epidemics are likely to be complete. Applications include a reduction of the cost of computing a semilinear predicate to $O(n \log^5 n)$ interactions from the previously best-known bound of $O(n^2 \log n)$ interactions and simulation of a LOGSPACE Turing machine using $O(n \log^2 n)$ interactions per step after an initial $O(n \log^{O(1)} n)$-interaction startup phase. These bounds on interactions translate into polylogarithmic time per step in a natural parallel model in which each agent participates in an expected $\Theta(1)$ interactions per time unit. Open problems are discussed, together with simulation results that suggest the possibility of removing the initial-leader assumption.

An extended abstract of this paper previously appeared in DISC 2006 [5]. Some additional material previously appeared in DISC 2007 [7]. The second author was supported in part by NSF grants CNS-0305258 and CNS-0435201.

Dana Angluin · James Aspnes
Department of Computer Science, Yale University

David Eisenstat
New Haven, Connecticut

## 1 Introduction

The **population protocol** model of Angluin *et al.* [3] consists of a population of finite-state agents that interact in pairs, where each interaction updates the state of both participants according to a transition function based on the participants' previous states and the goal is to have all agents eventually converge to a common output value that represents the result of the computation, typically a predicate on the initial state of the population. A population protocol that always converges to the correct output is said to perform **stable computation** and a predicate that can be so computed is called **stably computable**.

The model is motivated by systems of highly-restricted computational agents with little control over their movements, as in a chemical solution or a sensor network where sensors are attached to highly mobile objects (e.g., vehicles or animals). In this latter case, the predicate computed by the population will be some useful global property of the combination of sensor inputs.

In the simplest version of the model, any pair of agents may interact, but which interaction occurs at each step is under the control of an adversary, subject to a fairness condition that essentially says that any continuously reachable global configuration is eventually reached. The class of stably computable predicates in this model is now very well understood: it consists precisely of the **semilinear predicates** (those predicates on counts of input agents definable in first-order **Presburger arithmetic** [28]), where semilinearity was shown to be sufficient in [3] and necessary in [6, 10]. However, the fact that a protocol will eventually converge to the correct value of a semilinear predicate says little about how long such convergence will take.

Our fundamental measure of convergence is the total number of pairwise interactions until all agents have the correct output value, considered as a function of $n$, the number of agents in the population. We may also consider models in which reactions occur in parallel according to a Poisson process (as assumed in e.g. [21, 22]); this gives an equivalent distribution over sequences of reactions but suggests a mea-

sure of **parallel time** based on assuming each agent participates in an expected $\Theta(1)$ interactions per time unit. It is not hard to see that this time measure is asymptotically equal to the number of interactions divided by $n$.

To bound these measures, it is necessary to place further restrictions on the adversary: a merely fair adversary may wait an arbitrary number of interactions before it allows a particular important interaction to occur. In the present work, we consider the natural probabilistic model, proposed in [3], in which each interaction occurs between a pair of agents chosen uniformly at random. In this model, it was shown in [3] that any semilinear predicate can be computed in $O(n^2 \log n)$ expected interactions using a protocol based on leader election in which the leader communicates the outcome by interacting with every other agent; the large number of interactions is caused by the delay until the last surviving leader directly meets every other agent at least once. Protocols were also given to simulate randomized LOGSPACE computations with polynomial slowdown, allowing an inverse polynomial probability of failure.

We give a new method for the design of probabilistic population protocols, based on controlled use of self-timed epidemics to disseminate control information rapidly through the population. This method organizes a population as an array of registers that can hold values linear in the size of the population, stored in unary. The simulated registers support the usual arithmetic operations, including addition, subtraction, multiplication and division by constants, and comparison, with implementations that complete with high probability in $O(n \log^5 n)$ interactions and polylogarithmic parallel time per operation. As a consequence, any semilinear predicate can be computed without error by a probabilistic population protocol that converges in $O(n \log^5 n)$ interactions with high probability, and randomized LOGSPACE computation can be simulated with inverse polynomial error with only polylogarithmic slowdown. These bounds are optimal up to polylogarithmic factors, because $\Omega(n \log n)$ interactions are necessary to ensure that every agent has participated in at least one interaction with high probability. We also incorporate results of [7] that show that combining the approximate majority protocol that appeared there with a more sophisticated register representation can reduce many of these bounds to $O(n \log^2 n)$, at the cost of an initialization step that takes polylogarithmic parallel time.

However, in order to achieve these low running times, it is necessary to assume a leader in the form of some unique input agent. This is a reasonable assumption in sensor network models as a typical sensor network will have some small number of sensors that perform the specialized task of communicating with the user and we can appoint one of these as leader. Assuming the existence of a leader does not trivialize the problem; for example, any protocol that requires that the leader personally visit every agent in the population runs in expected number of at least $\Omega(n^2 \log n)$ interactions.

If a leader is not provided, it is in principle possible to elect one; however, the best known expected bounds for leader election in a population protocol is still the $\Theta(n^2)$ interactions or $\Theta(n)$ parallel time of a naive protocol in which candidate leaders drop out only on encountering other leaders. It is an open problem whether a leader can be elected significantly faster. We present simulation results for a complex leader election protocol that suggest that faster leader election is possible, but better techniques are needed to prove correctness of this protocol.

In building a register machine from agents in a population protocol, we must solve many of the same problems as hardware designers building register machines from electrons. Thus the structure of the paper roughly follows the design of increasing layers of abstraction in a CPU. We present the underlying physics of the world—the population protocol model—in Section 2. Section 3 gives concentration bounds on the number of interactions to propagate the epidemics that take the place of electrical signals and describes the phase clock used to coordinate the virtual machine's instruction cycle. Section 4 describes the microcode level of our machine, showing how to implement operations that are convenient to implement but hard to program with. More traditional register machine operations are then built on top of these microcode operations in Section 5, culminating in a summary of our main construction in Theorem 2. The further optimizations that originally appeared in [7] are described in Section 6. Applications to simulating LOGSPACE Turing machines and computing semilinear predicates are described in Section 7. The fast leader election protocol and an improved phase clock, along with simulation results for these protocols, are described in Section 8. Some directions for future work are described in Section 9.

Many of our results are probabilistic, and our algorithms include tuning parameters that can be used to adjust the probability of success. For example, the algorithm that implements a given register machine program is designed to run for $n^k$ instructions for some $k$, and the probability of failure for each instruction must be bounded by a suitable inverse polynomial in $n$. We say that a statement holds **with high probability** if for any constant $c$ there is a setting of the tuning parameters that cause the statement to hold with probability at least $1 - n^{-c}$. The cost of achieving a larger value of $c$ is a constant factor slowdown in the number of interactions (or time) used by the algorithms.

### 1.1 Related work

The population protocol model has been the subject of several recent papers. Diamadi and Fischer introduced a version of the probabilistic model to study the propagation of trust in a social network [19], and a related model of urn automata was explored in [2]. One motivation for the basic model studied in [3] was to understand the computational capabilities of populations of passively mobile sensors with very limited computational power. In the simplest form of the model, any agent may interact with any other, but variations of the model include limits on which pairs of agents

may interact [1,3,4], various forms of one-way and delayed communication [9,10], and failures of agents [18]. The properties computed by population protocols have also been extended from predicates on the initial population to predicates on the underlying interaction graph [1], self-stabilizing behaviors [11], and stabilizing consensus [12].

Similar systems of pairwise interaction have previously been used to model the interaction of small molecules in solution [22, 23] and the propagation in a human population of rumors [16] or epidemics of infectious disease [14]. One distinction in the literature on epidemics is whether individuals are removed (for example, by quarantine) from the population after they become infectious, thus becoming neither infectious nor susceptible to infection. The epidemics we construct are of the simple type, in which an infectious agent remains infectious until it receives a control signal to the contrary.

Birman *et al.* [15] introduced, analyzed and implemented a probabilistic bimodal multicast protocol based on gossip protocols and the mathematics of epidemics. In the simplest form of the protocol, in each of $R$ rounds, each process that receives a gossip message for the first time sends it to a random subset of other processes (chosen by flipping a coin of bias $\beta$ for each other process) and then removes itself from the protocol. The full analysis considers process and message delivery failures, and gives a recursive method of calculating a bound on the probability of failure of the multicast.

The notion of a "phase clock" as used in our protocol is common in the self-stabilizing literature, e.g. [25]. There is a substantial stream of research on building self-stabilizing synchronized clocks dating back to to the work of Arora *et al.* [13]. Recent work such as [20] shows that it is possible to perform self-stabilizing clock synchronization in traditional distributed systems even with a constant fraction of Byzantine faults; however, the resulting algorithms require more network structure and computational capacity at each agent than is available in a population protocol. An intriguing protocol of Daliot *et al.* [17] constructs a protocol for the closely-related problem of pulse synchronization inspired directly by biological models. Though this protocol also exceeds the finite-state limits of population protocols, it may be possible to construct a useful phase clock for our model by adapting similar techniques.

## 2 Model

In this paper we consider only the complete all-pairs interaction graph, so we can simplify the general definition of a probabilistic population protocol as follows. A **population protocol** consists of a finite set $Q$ of states, of which a nonempty subset $X$ are the initial states (thought of as inputs), a deterministic transition function $(a,b) \mapsto (a',b')$ that maps ordered pairs of states to ordered pairs of states, and an output function that maps states to an output alphabet $Y$. The **population** consists of agents numbered 1 through $n$; agent identities are not visible to the agents themselves, but facilitate the description of the model. A **configuration** $C$ is a map from the population to states, giving the current state of every agent. An **input configuration** is a map from the population to $X$, representing an input consisting of a multiset of elements of $X$. $C$ can reach $C'$ in one interaction, denoted $C \to C'$, if there exist distinct agents $i$ and $j$ such that $C(i) = a, C(j) = b$, the transition function specifies $(a,b) \mapsto (a',b')$ and $C'(i) = a'$, $C'(j) = b'$ and $C'(k) = C(k)$ for all $k$ other than $i$ and $j$. In this interaction, $i$ is the **initiator** and $j$ is the **responder** – this asymmetry of roles is an assumption of the model [4].

An **execution** is a sequence $C_1, C_2, \ldots$ of configurations such that for each $i$, $C_i \to C_{i+1}$. An execution **converges** to an output $y \in Y$, if there exists an $i$ such that for every $j \geq i$, the output function applied to every state occurring in $C_j$ is $y$. In general, individual agents may not know when convergence to a common output has been reached, and protocols are generally designed not to halt. An execution is **fair** if for any $C_i$ and $C_j$ such that $C_i \to C_j$ and $C_i$ occurs infinitely often in the execution, $C_j$ also occurs infinitely often in the execution. A protocol **stably computes** a predicate $P$ on multisets of elements of $X$ if for any input configuration $C$, every fair execution of the protocol starting with $C$ converges to 1 if $P$ is true on the multiset of inputs represented by $C$, and converges to 0 otherwise. Note that a fixed protocol must be able to handle populations of arbitrary finite size – there is no dependence of the number of states on $n$, the population size.

For a **probabilistic population protocol**, we stipulate a particular probability distribution over executions from a given configuration $C_1$ as follows. We generate $C_{k+1}$ from $C_k$ by drawing an ordered pair $(i, j)$ of agents independently and uniformly, applying the transition function to $(C_k(i), C_k(j))$, and updating the states of $i$ and $j$ accordingly to obtain $C_{k+1}$. (Note that an execution generated this way will be fair with probability 1.) In the probabilistic model we consider both the random variable of the number of interactions until convergence and the probabilities of various error conditions in our algorithms.

## 3 Tools

Here we give the basic tools used to construct our virtual machine. These consist of concentration bounds on the number of interactions needed to spread epidemics through the population (Section 3.1), which are then used to construct a phase clock that controls the machine's instruction cycle (Section 3.2). Basic protocols for duplication (Section 3.3), cancellation (Section 3.4), and probing (Section 3.5) are then defined and analyzed.

### 3.1 Epidemics

By a **one-way epidemic** we denote the population protocol with state space $\{0,1\}$ and transition rule $(x,y) \mapsto$

$(x, \max(x,y))$. Interpreting 0 as "susceptible" and 1 as "infected," this protocol corresponds to a simple epidemic in which transmission of the infection occurs if and only if the initiator is infected and the responder is susceptible. We wish to show that the number of interactions for the epidemic to finish (that is, infect every agent) is $\Theta(n \log n)$ with high probability.

To do so, we reduce the number of interactions of the epidemic protocol to the number of operations in the well-known coupon collector problem, in which balls are thrown uniformly at random into bins until every bin contains at least one ball. We show the following bounds on the number of operations to fill the last $k$ of $n$ bins based on an occupancy bound of Kamath *et al.* [26]. Because of the high variance associated with filling the last few bins, we consider only $k \geq n^{\varepsilon}$ for $\varepsilon > 0$.

**Lemma 1** *Let $S(k,n)$ be the number of operations to fill the last $k$ of $n$ bins in the coupon collector problem. Then for any fixed $\varepsilon > 0$ and $c > 0$, there exist positive constants $c_1$ and $c_2$ such that for all sufficiently large $n$ and any $k > n^{\varepsilon}$, $c_1 n \ln k \leq S(k,n) \leq c_2 n \ln k$ with probability at least $1 - n^{-c}$.*

*Proof* Observe that each step of collecting a specific $k$ of $n$ coupons can be split into (a) choosing to pick one of the $k$ coupons with probability $k/n$ and (if (a) is successful) (b) choosing the specific coupon to pick. The number of steps of type (b) before all coupons are collected is exactly $S(k,k)$. It is easy to see that $E[S(k,n)] = \frac{n}{k} E[S(k,k)]$ and a simple application of Chernoff bounds shows that $S(k,n) = \Theta(\frac{n}{k} S(k,k))$ with high probability (assuming $k$ is polynomial in $n$).

We will now show that $S(k,k) = \Omega(k \log k)$ with high probability. Theorem 2 of [26] states that with $m$ balls tossed uniformly into $n$ bins,

$$\Pr[|Z - \mu| \geq \rho\mu] \leq 2 \exp\left( -\frac{\rho^2 \mu^2 (n - \frac{1}{2})}{n^2 - \mu^2} \right), \tag{1}$$

where $Z$ is the number of empty bins and $\mu = E[Z] = n(1 - 1/n)^m = \Theta(ne^{-m/n})$.

Our goal is to bound the probability that $Z = 0$, i.e. that all coupons have been collected after $m$ operations. Substitute $n = k$ and $m = (1/4)k \ln k$ in (1). This gives $\mu = \Theta(ke^{-(1/4)\ln k}) = \Theta(k \cdot k^{-1/4}) = \Theta(k^{3/4})$. For $Z$ to equal 0 we must have $|Z - \mu| \geq \mu$ or $\rho = 1$. So

$$\begin{aligned} \Pr[Z = 0] &\leq 2 \exp\left( -\frac{\mu^2 (k - \frac{1}{2})}{k^2 - \mu^2} \right) \\ &= 2 \exp\left( -\Theta\left( \frac{k^{3/2} k}{k^2} \right) \right) \\ &= 2 \exp\left( -\Theta\left( k^{1/2} \right) \right). \end{aligned}$$

For $k$ polynomial in $n$ we get an exponentially small probability that $S(k,k) \leq \frac{1}{4} k \ln k$, which tells us that $S(k, n-1) = \Omega(\frac{n-1}{k} \cdot k \ln k) = \Omega(n \ln k)$ with high probability.

For the upper bound, if $m = ak \ln k$ and $n = k$ then $E[Z] = k(1 - 1/k)^{ak \ln k} \leq ke^{-a \ln k} = k^{1-a}$ can be made an arbitrarily small polynomial in $n$ by choosing $a$ sufficiently large, and by Markov's inequality this bounds $\Pr[Z \geq 1]$.

Let $T(k)$ denote the number of interactions to infect the last $k$ agents in a one-way epidemic. To get concentration bounds for $T(k)$, we now reduce the number of interactions for an epidemic to the number of operations for coupon collector. The intuition is that once half the agents are infected, we have a coupon collector problem (collecting susceptible responders) that is delayed by at most a factor of two by occasionally choosing uninfected initiators. When fewer than half the agents are infected, we use the symmetry of the waiting times for each new infection to apply the same bounds.

It will be useful to have a slightly more general lemma that bounds the time to infect the first $k$ susceptible agents. Because of the high variance associated with filling the last few bins in the coupon collection problem, we consider only $k \geq n^{\varepsilon}$ for $\varepsilon > 0$.

**Lemma 2** *Let $T(k)$ be number of interactions before a one-way epidemic starting with a single infected agent infects $k$ agents. For any fixed $\varepsilon > 0$ and $c > 0$, there exist positive constants $c_1$ and $c_2$ such that for sufficiently large $n$ and any $k > n^{\varepsilon}$, $c_1 n \ln k \leq T(k) \leq c_2 n \ln k$ with probability at least $1 - n^{-c}$.*

*Proof* We start by observing some useful symmetry properties. The probability that an interaction produces $i+1$ infected nodes starting from $i$ infected nodes, which is $\frac{i(n-i)}{n(n-1)}$, is also the probability that an interaction produces $n-i+1$ infected nodes starting from $n-i$ infected nodes. It follows that the distribution of $T(i+1) - T(i)$ is identical to the distribution of $T(n-i+1) - T(n-i)$ and in general that the distribution of $T(k) = T(k) - T(1)$—the number of interactions to infect the first $k-1$ susceptibles—is identical to that of $T(n) - T(n-k+1)$—the number of interactions to infect the last $k-1$.

Next we'll bound the number of interactions to infect the last $k$ susceptibles using Lemma 1. Consider each step of the epidemic process as consisting of (a) a random choice of whether or not the initiator is infected and (b) a random choice of a responder. Then step (b) makes progress with probability $k/(n-1)$, exactly the same probability as in the coupon collector problem with $k$ remaining coupons out of $n-1$. Step (a) corresponds to a random choice (with probability $(n-k)/n$ lying between $1/2$ and 1) of whether to draw a coupon or not. A straightforward Chernoff bound argument applied to (a) then shows that the number of interactions to infect the last $k$ susceptibles lies with high probability between $S(k, n-1)$ and $2S(k, n-1)$ where $S(k, n-1)$ is the time to collect the last $k$ of $n-1$ coupons. From Lemma 1, we have that $S(k, n-1)$ lies between $c_1(n-1)\ln k$ and $c_2(n-1)\ln k$ with high probability, which simplifies to $\Theta(n \ln k)$ as claimed.

### 3.2 The phase clock

The core of our construction is a **phase clock** that allows a leader to determine when an epidemic or sequence of triggered epidemics is likely to have finished. In essence, the phase clock allows a finite-state leader to count off $\Theta(n \log n)$ total interactions with high probability; by adjusting the constants in the clock, the resulting count is enough to outlast the $c_2 n \ln n$ interactions needed to complete an epidemic by Lemma 2. Like physical clocks, the phase clock is based on a readily-available natural phenomenon with the right duration constant. A good choice for this natural phenomenon, in a probabilistic population protocol, turns out to be itself the spread of an epidemic. Like the one-way epidemic of Section 3.1, the phase clock requires only one-way communication.

Here is the protocol: each agent has a state in the range $0 \ldots m-1$ for some constant $m$ that indicates which phase of the clock it is infected with. (The value of $m$ will be chosen independent of $n$, but depending on $c$, where $1-n^{-c}$ is the desired success probability.) Up to a point, later phases overwrite earlier phases: a responder in phase $i$ will adopt the phase of any initiator in phases $i+1 \bmod m$ through $i+m/2 \bmod m$, but will ignore initiators in other phases. This behavior completely describes the transition function for non-leader responders.

New phases are triggered by a unique leader agent. When the leader encounters an initiator with its own phase, it spontaneously moves to the next phase. The leader ignores interactions with initiators in other phases. The initial configuration of the phase clock has the leader in phase 0 and all other agents in phase $m-1$. A **round** consists of $m$ phases. A new round starts when the leader enters phase 0.

The normal operation of the phase clock has all the agents in a very few adjacent states, with the leader in the foremost one. When that state becomes populated enough for the leader to encounter another agent in that state, the leader moves on to the next state (modulo $m$) and the followers are pulled along. Successive rounds should be $\Theta(n \log n)$ interactions apart with high probability; the lower bound allows messages sent epidemically to reach the whole population, and the upper bound is essential for the overall efficiency of our algorithms.

#### *3.2.1 Analysis*

We wish to show that for appropriate constants $c$ and $m$, any epidemic (running in parallel with the phase clock) that starts in phase $i$ completes by the next occurrence of phase $(i+c) \mod m$ with high probability. To simplify the argument, we first consider an infinite-state version of the phase clock with state space $\mathbb{Z} \times \{\text{leader}, \text{follower}\}$ and transition rules

$$(x,b),(y,\text{follower}) \mapsto (x,b),(\max(x,y),\text{follower})$$
$$(x,b),(x,\text{leader}) \mapsto (x,b),(x+1,\text{leader})$$
$$(x,b),(y,\text{leader}) \mapsto (x,b),(y,\text{leader}) \quad [y \neq x]$$

We assume the initial configuration (at interaction 0) has the leader in state 0 and each follower in state $-1$. This infinite-state protocol has the useful invariant that every agent has a phase less than or equal to that of the leader. We define phase $i$ as starting when the leader agent first adopts phase $i$. This result bounds the probability that a phase "ends too early" by $n^{-1/2}$.

**Lemma 3** *Let phase $i$ start at interaction $t$. Then there is a constant $a$ such that for sufficiently large $n$, phase $i+1$ starts before interaction $t + an \ln n$ with probability at most $n^{-1/2}$.*

*Proof* Call an agent infected if and only if its phase is at least $i$. At the start of phase $i$, we have exactly one infected agent, and an examination of the infinite-state transition rule reveals that infection spreads as in a standard one-way epidemic.

We wish to bound the probability that the leader encounters an infected initiator by interaction $t + an \ln n$. We can do so using the following bit of trickery: consider an alternative version of the epidemic where at the start of phase $i$ it is not the leader who is infected but a follower. In this modified epidemic the distribution on the number of infected agents after each interaction is identical to the original protocol, but as there is one more infected follower the probability that the leader encounters an infected agent is only increased and thus the probability that the leader has not encountered an infected follower by interaction $t + an \ln n$ has decreased. But we can easily detect in the modified epidemic whether the leader has encountered an infected follower: it has done so if and only if it is infected itself.

Now observe from Lemma 2 that there is a constant $c_1$ such that the probability that more than $n^{1/4}$ (say) agents are infected by interaction $t + c_1 n \ln n^{1/4} = t + (c_1/4) n \ln n$ is at most $n^{-1}$. By symmetry, if only $n^{1/4}$ agents are infected, the probability that the leader is one of this agents is bounded by $n^{-3/4}$. So the probability that the leader is infected at interaction $t + an \ln n$ when $a = c_1/4$ is at most $n^{-1} + n^{-3/4} \leq n^{-1/2}$.

Observing that several phases must "end too early" in order for a round to "end too early" allows us to go from a failure probability of $n^{-1/2}$ for a phase to $n^{-c}$ for a round.

**Corollary 1** *Let phase $i$ start at interaction $t$. Then for any $c > 0$ and $d > 0$, there is a constant $k$ such that for sufficiently large $n$, phase $i+k$ starts before $t + dn \ln n$ interactions with probability at most $n^{-c}$.*

*Proof* Call phase $j$ **short** if it finishes within $an \ln n$ interactions, where $a$ is as in Lemma 3. For any fixed odd $k$, the probability that more than half of the $k$ phases $i$ through $i+k-1$ are short is bounded by $2^{k-1}(n^{-1/2})^{(k+1)/2} = 2^{k-1} n^{-(k+1)/4} \leq n^{-k/8}$ when $n > 2^8$.

If fewer than half the phases are short, the other $k/2$ or more phases must take at least $an \ln n$ interactions each, for a total of at least $a(k/2) n \ln n$ interactions. Setting $k = \max(8c, 2d/a)$ then gives the claimed bound.

The following theorem gives probabilistic guarantees for a polynomial number of rounds of the phase clock. In the proof the probability of failure due to a "straggler" (agent so far behind that it appears to be ahead modulo $m$) must be also be appropriately bounded, to ensure that $m$ may be a constant independent of $n$.

**Theorem 1** *For any fixed $c, d_1 > 0$, there exist constants $m$ and $d_2$ such that, for all sufficiently large $n$, with probability at least $1 - n^{-c}$ the finite-state phase clock with parameter $m$, starting from an initial state consisting of one leader in phase 0 and $n-1$ followers in phase $m-1$, completes $n^c$ rounds of $m$ phases each, where the minimum number of interactions in any of the $n^c$ rounds is at least $d_1 n \ln n$ and the maximum is at most $d_2 n \ln n$,*

*Proof* The essential idea of the lower bound is to apply Corollary 1 twice: once to show that with high probability the number of interactions between phase $i+1$ and phase $i + m/2$ is long enough for any old phase-$i$ agents to be eaten up (thus avoiding any problems with wrap-around), and once to show the lower bound on the length of a round.

To show that no agent is left behind, consider, in the infinite-state protocol, the fate of agents in phase $i$ or lower once at least one agent in phase $i+1$ or higher exists. If we map all phases $i$ or lower to 0 and all phases $i+1$ or higher to 1, then encounters between agents have the same effect after the mapping as in a one-way epidemic. By Lemma 2, there is a constant $c_2$ such that all $n$ agents are infected by interaction $c_2 n \ln n$ with probability at least $1 - n^{-3c}$. By Corollary 1, there is a constant $k_1$ such that phase $i + k_1 + 1$ starts at least $c_2 n \ln n$ interactions after phase $i+1$ with probability at least $1 - n^{-3c}$. Setting $m > 2(k_1 + 1)$ then ensures that all phase $i$ (or lower) agents have updated their phase before phase $i + m/2$ with probability at least $1 - 2n^{-3c}$. If we sum the probability of failure over all $mn^c$ phases in the first $n^c$ rounds, we get a probability of at most $2mn^{-2c}$ that some phase $i$ agent survives long enough to cause trouble.

Assuming that no such trouble occurs, we can simulate the finite-state phase clock by mapping the phases of the infinite-state phase clock mod $m$. Now by Corollary 1 there is a constant $k_2$ such that the number of interactions to complete $k_2$ consecutive phases is at least $d_1 n \ln n$ with probability at least $1 - n^{-3c}$. Setting $m \geq k_2$ thus gives that all $n^c$ rounds take at least $d_1 n \ln n$ interactions with probability at least $1 - n^c n^{-3c} = 1 - n^{-2c}$. Thus the total probability of failure on the lower bound side is bounded by $2mn^{-2c} + n^{-2c}$.

For the upper bound, choose $c_2$ in Lemma 2 so that each epidemic completes at most $c_2 n \ln n$ interactions with probability at least $1 - n^{-3c}$. If this condition holds for all $m$ phases of a round, then the round completes in at most $d_1 n \ln n$ interactions where $d_1 = mc_2$. The probability that this bound fails, summed over all $mn^c$ phases, is at most $mn^{-2c}$.

The total probability of failure on both the lower and upper bound side is thus bounded by $3mn^{-2c} + n^{-2c} < n^{-c}$ for sufficiently large $n$.

### 3.3 Duplication

A **duplication** protocol has state space $\{(1,1),(0,1),(0,0)\}$ and transition rules:

$(1,1),(0,0) \mapsto (0,1),(0,1)$
$(0,0),(1,1) \mapsto (0,1),(0,1)$

with all other encounters having no effect.

When run to convergence, a duplication protocol starting with $a$ "active" agents in state $(1,1)$ and the rest in the null state $(0,0)$ converges to $2a$ "inactive" agents in state $(0,1)$, provided $2a$ is less than $n$; otherwise it converges to a population of mixed active and inactive agents with no unrecruited agents left in the null state. The invariant is that the total number of 1 tokens is preserved while eliminating as many double-token agents as possible. We do not consider agents in a $(1,0)$ state as they can be converted to $(0,1)$ immediately at the start of the protocol.

When the initial number of active agents $a$ is close to $n/2$, duplication may take as much as $\Theta(n^2)$ interactions to converge, as the last few active agents wait to encounter the last few null agents. But for smaller values of $a$ the protocol converges more quickly.

**Lemma 4** *Let $2a + b \leq n/2$. The probability that a duplication protocol starting with a active agents and b inactive agents, has not converged after $(2c+1)n \ln n$ interactions is at most $n^{-c}$.*

*Proof* Note that $2a + b$ is the number of agents that will eventually be in a non-null state if the protocol converges. Under the assumption that $2a + b \leq n/2$, at least half the agents remain in the null state throughout the protocol. So the probability that a particular active agent is selected as initiator and encounters a null responder on any given interaction is at least $\frac{1}{2n}$. The probability that it never becomes inactive in $2(c+1)n \ln n$ interactions is thus $(1 - 1/2n)^{2(c+1)n \ln n} \leq \exp\left(-\frac{2(c+1)n \ln n}{2n}\right) = n^{-c-1}$. The expected number of surviving active agents after $2(c+1)n \ln n$ steps is thus at most $an^{-c-1} < n^{-c}$ and the full result follows from Markov's inequality.

### 3.4 Cancellation

A **cancellation** protocol has states $\{(0,0),(1,0),(0,1)\}$ and transition rules:

$(1,0),(0,1) \mapsto (0,0),(0,0)$
$(0,1),(1,0) \mapsto (0,0),(0,0)$

It maintains the invariant that the number of 1 tokens in the left-hand position minus the number of 1 tokens in the right-hand position is fixed. It converges when only $(1,0)$ and

$(0,0)$ or only $(0,1)$ and $(0,0)$ agents remain. We assume that there are no $(1,1)$ agents as these can be converted to $(0,0)$ agents at the start of the protocol. We refer to agents in state $(1,0)$ or $(0,1)$ as nonzero agents.

As with duplication, the number of interactions to converge when $(1,0)$ and $(0,1)$ are nearly equally balanced can be as many as $\Theta(n^2)$, since we must wait in the end for the last few survivors to find each other. This is too slow to use cancellation to implement subtraction directly. Instead, we will use cancellation for inequality testing, using duplication to ensure that there is a large enough majority of one value or the other to ensure fast convergence. We will use the following fact.

**Lemma 5** *Starting from any initial configuration, with probability at least $1-n^{-c}$, after $4(c+1)n\ln n$ interactions a cancellation protocol has either converged or has at most $n/8$ of each type of nonzero agent.*

*Proof* Assume without loss of generality that there are at least as many $(1,0)$ agents as $(0,1)$ agents. Pick a particular $(0,1)$ agent from the initial population and for each $t$ let $A_t$ be the event that after $t$ interactions this agent is still in state $(0,1)$ and there are at least $n/8$ agents in state $(1,0)$.

We will show by induction on $t$ that $\Pr[A_t] \le \left(1-\frac{1}{4n}\right)^t$. The base case is trivial. For the induction step, we have $\Pr[A_t] = \Pr[A_t|A_{t-1}]\Pr[A_{t-1}] + \Pr[A_t|A_{t-1}=0]\Pr[A_{t-1}=0] = \Pr[A_t|A_{t-1}]\Pr[A_{t-1}] \le \Pr[A_t|A_{t-1}]\left(1-\frac{1}{4n}\right)^{t-1}$.

Now let us bound $\Pr[A_t|A_{t-1}]$. Since $A_{t-1}$ occurs, after $t-1$ interactions our target $(0,1)$ agent has not yet encountered a $(1,0)$ agent and there are still at least $n/8$ surviving $(1,0)$ agents. The probability that the target agent and a $(1,0)$ agent are selected in interaction $t$ is thus at least $\frac{1}{4n}$. Thus there is a probability of at most $\left(1-\frac{1}{4n}\right)$ that the target agent survives, and we have

$$\begin{aligned}\Pr[A_t] &\le \Pr[A_t|A_{t-1}]\left(1-\frac{1}{4n}\right)^{t-1} \\ &\le \left(1-\frac{1}{4n}\right)^{t}.\end{aligned}$$

Setting $t = 4(c+1)n\ln n$ gives

$$\begin{aligned}\Pr[A_t] &\le \left(1-\frac{1}{4n}\right)^{4(c+1)n\ln n} \\ &\le \exp\left(-\frac{4(c+1)n\ln n}{4n}\right) \\ &= n^{-c-1}.\end{aligned}$$

Taking a union bound over all $(0,1)$ agents in the original population shows that with probability at least $1-n^{-c}$ either every $(0,1)$ agent is gone or there are at most $n/8$ $(1,0)$ agents. In the latter case, there are also at most $n/8$ $(0,1)$ agents by the assumption that $(1,0)$ was not initially in the minority, and this condition is preserved by the protocol.

### 3.5 Probing

A **probing** protocol is used to detect if any agents satisfying a given predicate exist. It uses three states 0, 1, and 2 (in addition to any state tested by the predicate) and has transition rules

$$(x,y) \mapsto (x, \max(x,y))$$

when the responder does not satisfy the predicate and

$$\begin{aligned}(0,y) &\mapsto (0,y) \\ (x,y) &\mapsto (x,2) \quad [x>0]\end{aligned}$$

when the responder does. Note that this is a one-way protocol.

To initiate a probe, a leader starts in state 1; this state spreads through an initial population of state 0 agents as in a one-way epidemic and triggers the epidemic spread of state 2 if it reaches an agent that satisfies the predicate.

**Lemma 6** *For any $c>0$, there is a constant $d$ such that for sufficiently large $n$, with probability at least $1-n^{-c}$ it is the case that after $dn\ln n$ interactions in the probing protocol either (a) no agent satisfies the predicate and every agent is in state 1, or (b) some agent satisfies the predicate and every agent is in state 2.*

*Proof* For case (a), apply Lemma 2. For case (b), apply Lemma 2 first to show that some satisfying agent is reached and again to show that the resulting 2 epidemic spreads to all agents.

## 4 Computation by epidemic: the microcode level

In this section, we describe how to construct an abstract register machine on top of a population protocol. This machine has a constant number of registers each capable of holding integer values in the range 0 to $n$, and supports the usual arithmetic operations on these registers, including addition, subtraction, multiplication and division by constants, inequality tests, and so forth. Each of these operations takes at most a polylogarithmic number of basic instruction cycles, where an instruction cycle takes $\Theta(n\log n)$ interactions or $\Theta(\log n)$ parallel time.

The simulation is probabilistic; there is an inverse polynomial probability of error for each operation, on which the exponent can be made arbitrarily large at the cost of increasing the constant factor in the running time.

The value of each register is distributed across the population in unary. For each register $A$, every member $i$ of the population maintains one bit $A_i$ and the current value of $A$ is simply $\sum_i A_i$. Thus the finite state of each agent can be thought of as a finite set of finite-valued control variables, and one boolean variable for each of a finite set of registers. Recall that the identities of agents are invisible to the agents themselves, and are used to facilitate description of the model.

| Instruction | Effect on state of agent $i$ |
|---|---|
| NOOP | No effect. |
| SET$(A)$ | Set $A_i = 1$. |
| COPY$(A,B)$ | Copy $A_i$ to $B_i$ |
| DUP$(A,B)$ | Run duplication protocol on state $(A_i,B_i)$. |
| CANCEL$(A,B)$ | Run cancellation protocol on state $(A_i,B_i)$. |
| PROBE$(A)$ | Run probe protocol with predicate $A_i = 1$. |

**Table 1** Instructions at the microcode level.

We assume there is a leader agent that organizes the computation; part of the leader's state stores the finite-state control for the register machine. We make a distinction between the "microcode layer" of the machine, which uses the basic mechanisms of Section 3, and the "machine code" layer, which provides familiar arithmetic operations.

At the microcode layer, we implement a basic instruction cycle in which the leader broadcasts an instruction to all agents using an epidemic. The agents then carry out this instruction until stopped by a second broadcast from the leader. This process repeats until the computation terminates.

To track the current instruction, each agent (including the leader) has a **current instruction register** in addition to its other state. These instructions are tagged with a **round number** in the range $0,1,2$, where round $i$ instructions are overwritten by round $i+1 \pmod 3$ instructions.

The instructions and their effects are given in Table 1. Most take registers as arguments. We also allow any occurrence of a register to be replaced by its negation, in which case the operation applies to those agents in which the appropriate bit is not set. For example, SET$(\neg A)$ resets $A_i$, PROBE$(\neg A)$ tests for agents in which $A_i$ is not set, COPY$(\neg A,B)$ sets $B_i$ to the negation of $A_i$, and so forth.

To interpret the table entries: when an agent changes its current instruction register to SET$(A)$, it sets its boolean variable for register $A$ to 1 and waits for the next instruction. Similarly, when it changes its current instruction register to COPY$(A,B)$, then the agent sets its boolean variable for register $B$ to the value of its boolean variable for register $A$. When its current instruction becomes DUP$(A,B)$, then the agent begins running the duplication protocol (Section 3.3) on the ordered pair of its boolean variables for registers $A$ and $B$. (In the case of $(1,0)$, it immediately exchanges them to $(0,1)$, and in the cases of $(1,1)$ and $(0,0)$, it participates in the duplication protocol when it interacts with other agents with current instruction DUP$(A,B)$, until either its pair becomes inactive or a new instruction supersedes the current one.) CANCEL$(A,B)$ and PROBE$(A)$ are handled analogously, where the predicate probed is whether the agent's boolean variable for register $A$ is 1. We omit describing the underlying transitions as the details are straightforward but lengthy.

When the leader updates its own current instruction register, the new value spreads to all other agents in $\Theta(n\log n)$ interactions with high probability (Lemma 2). The NOOP, SET, and COPY operations take effect immediately, so no additional interactions are required. The PROBE operation may require waiting for a second triggered epidemic, but the total interactions are still bounded by $O(n\log n)$ with high probability (by Lemma 6). Only the DUP and CANCEL operations may take longer to converge. Because subsequent operations overwrite each agent's current instruction register, issuing a new operation has the effect of cutting these operations off early. But if this new operation is issued $\Omega(n\log n)$ interactions later, the DUP operation converges with high probability unless it must recruit more than half the agents (Lemma 4), and the CANCEL operation either converges or leaves at most $n/4$ uncanceled values (Lemma 5). Note that for either operation, which outcome occurred can be detected with COPY and PROBE operations.

Thus, the leader waits for $\Omega(n\log n)$ interactions between issuing successive instructions, where the constant is chosen based on the desired error bound. But this can be done using a phase clock with appropriate parameter (Theorem 1): if it is large enough that both the probability that an operation completes too late and the probability that some phase clock triggers too early is $o(n^{-2c})$ per operation, then the total probability that any of $n^c$ operations fails is $o(n^{-c})$.

## 5 Computation by epidemic: higher-level operations

The operations of the previous section are not very convenient for programming. In this section, we describe how to implement more traditional register operations.

These can be divided into two groups: those that require a constant number of microcode instructions, and those that are implemented using loops. The first group, shown in Table 2, includes assignment, addition, multiplication by a constant, and zero tests. The second group includes comparison (testing for $A < B$, $A = B$, or $A > B$), subtraction, and division by a constant (including obtaining the remainder). These operations are described in more detail below.

### 5.1 Comparison

For comparison, it is tempting just to apply CANCEL and see what tokens survive. But if the two registers $A$ and $B$ being compared are close in value, then CANCEL may take $\Theta(n^2)$ interactions to converge. Instead, we apply up to $2\lg n$ rounds of cancellation, alternating with duplication steps that double the discrepancy between $A$ and $B$. If $A > B$ or $B > A$, the difference soon becomes large enough that all of the minority tokens are eliminated. The case where $A = B$ is detected by failure to converge, using a counter variable $C$ that doubles every other round.

The algorithm is given as Algorithm 1. It uses registers $A', B'$, and $C$ plus a bit $r$ to detect even-numbered rounds.

**Lemma 7** *The COMPARE algorithm (Algorithm 1) returns the correct answer with high probability after executing at most $O(\log n)$ microcode operations.*

| Operation | Effect | Implementation | Notes |
|---|---|---|---|
| Constant 0 | $A \leftarrow 0$ | SET($\neg A$) | |
| Constant 1 | $A \leftarrow 1$ | SET($\neg A$)<br>$A_{\text{leader}} \leftarrow 1$ | |
| Assignment | $A \leftarrow B$ | COPY($B$,$A$) | |
| Addition | $A \leftarrow A+B$ | COPY($B$,$X$)<br>DUP($X$,$A$)<br>PROBE($X$) | May fail with $X \neq 0$ if $A+B > n/2$. |
| Multiplication | $A \leftarrow kB$ | Use repeated addition. | $k = O(1)$ |
| Zero test | $A \neq 0$? | PROBE($A$) | |

**Table 2** Simple high-level operations and their implementations. Register $X$ is an auxiliary register.

**Algorithm 1** Comparison algorithm COMPARE.

```
 1: A' ← A.
 2: B' ← B.
 3: C ← 1.
 4: r ← 0.
 5: while true do
 6:     CANCEL(A', B').
 7:     if A' = 0 and B' = 0 then
 8:         return A = B.
 9:     else if A' = 0 then
10:         return A < B.
11:     else if B' = 0 then
12:         return A > B.
13:     end if
14:     r ← 1 − r.
15:     if r = 0 then
16:         C ← C + C.
17:         if addition failed then
18:             return A = B.
19:         end if
20:     end if
21:     A' ← A' + A'.
22:     B' ← B' + B'.
23: end while
```

*Proof* Observe that $A'-B'$ is initially equal to $A-B$, and that the value of $A'-B'$ is preserved by the CANCEL operation in step 6. By Lemma 5, if the loop does not terminate with $A'$ or $B'$ equal to zero, both registers hold values of at most $n/8$. It follows that the addition operations in steps 21 and 22 succeed with high probability, leaving $A'-B' = 2^i(A-B)$ after $i$ such doublings. In particular, after $\lceil \lg n \rceil$ doublings, $|A'-B'| \geq n|A-B|$, a contradiction unless $A = B$. So if the protocol does not terminate before $\lceil \lg n \rceil$ passes through the loop, we have $A = B$, which is eventually detected after at most $2\lg n$ passes when $C$ grows too large. Since there are only $O(1)$ microcode operations per iteration, the claim follows.

## 5.2 Subtraction

Subtraction is the inverse of addition, and addition is a monotone operation. It follows that we can implement subtraction using binary search. The cost of this implementation is high ($O(\log^3 n)$ microcode operations per subtraction), but it is the best we know how to do with a simple unary register representation. With a more sophisticated representation, the cost is no higher than addition (see Section 6.2 below).

Our rather rococo algorithm for computing $C \leftarrow A-B$, given as Algorithm 2, repeatedly looks for the largest power of two that can be added to the candidate difference $C$ without making the sum of the difference $C$ and the subtrahend $B$ greater than the minuend $A$. It obtains one more 1 bit of the difference for each iteration.

The algorithm assumes $A \geq B$. An initial cancellation step is used to handle particularly large inputs. This allows the algorithm to work even when $A$ lies outside the safe range of the addition operation.

The algorithm uses several auxiliary registers to keep track of the power of two to add to $C$ (this is the $D$ register) and to perform various implicit sums and tests (as in computing $B'+C+D+D$).

**Algorithm 2** Subtraction algorithm SUBTRACT.

```
 1: A' ← A.
 2: B' ← B.
 3: CANCEL(A', B').
 4: if B' = 0 then
 5:     C ← A.
 6:     return.
 7: end if
 8: C ← 0.
 9: while A' ≠ B' + C do
10:     D ← 1.
11:     while A' ≥ B' + C + D + D do
12:         D ← D + D.
13:     end while
14:     C ← C + D.
15: end while
```

**Lemma 8** *When $A \geq B$, the SUBTRACT algorithm (Algorithm 2) computes $C \leftarrow A-B$ with high probability in $O(\log^3 n)$ microcode operations.*

*Proof* By Lemma 5, after Line 3 either $B' = 0$ or $A' \leq n/8$ with high probability. In the former case the algorithm returns immediately, so we can safely assume $A' \leq n/8$ for the remainder. This also gives bounds of $n/8$ on $B'$, $C$, and $D$, so that the largest sum computed is $B'+C+D+D \leq n/2$, which lies within the safe addition range.

To show that the algorithm terminates as advertised, observe that each iteration of the outer loop sets $C \leftarrow C+D$

where $B'+C+D \leq A' < B'+C+2D$, from which $D$ is the largest power of two such that $C+D \leq A'-B' = A-B$. Thus each iteration of the outer loop adds a distinct 1 bit to $C$'s binary representation, and after $\lceil \lg n/8 \rceil$ such iterations there are no more bits to be added. Since the inner loop doubles $D$ each iteration, it also runs for at most $\lceil \lg n \rceil$ iterations each time it is invoked, for a total of $O(\log^2 n)$ lines executed. However, the comparison in Line 11 may take up to $O(\log n)$ microcode operations, so the total cost is $O(\log^3 n)$ operations as claimed.

### 5.3 Division

Division of $A$ by a constant $k$ is analogous to subtraction; we set $A' \leftarrow A$ and $B \leftarrow 0$ and repeatedly seek the largest power of two $D$ such that $kD$ can be successfully computed (i.e., does not cause addition to overflow) and $kD \leq A'$. We then subtract $kD$ from $A'$ and add $D$ to $B$.

The protocol terminates when $A' < k$, i.e. when no value of $D$ works. At this point $B$ holds the quotient $\lfloor A/k \rfloor$ and $A'$ the remainder $A$ mod $k$. Since each iteration adds one bit to the quotient, there are at most $O(\log n)$ iterations of the outer loop, for a total cost of $O(\log^4 n)$ microcode operations (since each outer loop iteration requires one subtraction operation). This yields a total cost for division of $O(n \log^5 n)$ interactions on average.[1]

One curious property of this protocol is that the leader does not learn the value of the remainder, even though it is small enough to fit in its limited memory. If it is important for the leader to learn the remainder, it can do so using $k$ addition and comparison operations, by successively testing the remainder $A'$ for equality with the values $0, 1, 1+1, 1+1+1, \ldots, k$. The cost of this test is dominated by the cost of the division algorithm.

### 5.4 Other operations

Multiplication and division by constants give us the ability to extract individual bits of a register value $A$. This is sufficient to implement basic operations like $A \leftarrow B \cdot C$, $A \leftarrow \lfloor B/C \rfloor$ in polylogarithmic time using standard bitwise algorithms.

### 5.5 Summary

Combining preceding results gives:

**Theorem 2** *A probabilistic population with an initial leader can simulate steps of a virtual machine with a constant number of registers holding integer values in the range* 0 *to n, where each step consists of (a) assigning a constant* 0 *or* 1 *value to a register; (b) assigning the value of one register to another; (c) adding the value of one register to another, provided the total does not exceed $n/2$; (d) multiplying a register by a constant, provided the result does not exceed $n/2$; (e) testing if a register is equal to zero; (f) comparing the values of two registers; (g) subtracting the values of two registers; or (h) dividing the value of a register by a constant and computing the remainder. The probability that for any single operation the simulation fails or takes more than $O(n \log^5 n)$ interactions can be made $O(n^{-c})$ for any fixed $c$.*

## 6 Further optimizations

By using a more sophisticated representation for registers together with our recent protocol for fast approximate majority [7], we can remove several of the logarithmic factors in the cost of our register machine construction.[2]

### 6.1 Faster comparison using approximate majority

A major bottleneck in our construction is the cost of comparison operations.

In a recent paper [7], we showed that a simple three-state algorithm computes the majority bit among its input bits in $O(n \log n)$ interactions with high probability, provided the initial majority is sufficiently large and all agents start the approximate majority protocol at the same time. An extended version of this result, found in the journal version of the paper [8], shows that the same result holds with an initial majority of $\Omega(n^{3/4+\varepsilon})$ if all but one agent is initially dormant and the rest are added to the protocol using an epidemic wake-up process. We can use this result to speed up comparisons in our register machine construction.

To apply this to our register machine, we change the register representation to ensure that a large enough gap between any two different register values exists. We guarantee this by having registers hold values that are multiples of $n^{4/5}$; five such registers are sufficient to represent $n = (n^{1/5})^5$ different values, thinking of them as five **wide-digits** of a number in base $n^{1/5}$. Thus, to compare two wide digits, say $A$ and $B$, we do an approximate majority comparison of $A + (1/2)n^{4/5}$ with $B$; if the result is that $A$ is in the majority, then we conclude that $B \leq A$, otherwise that $A < B$. To compare two registers composed of $O(1)$ wide-digits it suffices to proceed digit by digit. The actual digit comparison operation is a straightforward application of the fast robust approximate majority protocol, where the wake-up process is implemented by the epidemic instructing all agents to execute the comparison operation.

### 6.2 Faster subtraction using a balanced representation

The subtraction operation of Section 5.2 requires $O(\log n)$ rounds of binary search, where the $O(\log^2 n)$ parallel time comparison operation dominates the cost of each round.

[1] This value was incorrectly reported as $O(n \log^4 n)$ in [5].

[2] Much of the material in this section originally appeared in [7].

Though we could replace these comparisons with our faster comparison operation and reduce the cost of subtraction to $O(\log^2 n)$, we can obtain a still better reduction to $O(\log n)$ parallel time by use the logician's construction of the integers from the natural numbers: the value $A$ in a register is represented by the difference $A_+ - A_-$ of values in two different registers. To compute $C \leftarrow A + B$, we compute $C_+ \leftarrow A_+ + B_+$ and $C_- \leftarrow A_- + B_-$. To compute $C \leftarrow A - B$, we compute $C_+ \leftarrow A_+ + B_-$ and $C_- \leftarrow A_- + B_+$. These operations both take parallel time $O(\log n)$, because addition is already $O(\log n)$ in the previous construction. To keep the $+$ and $-$ parts of the registers from growing too large, we permit them to cancel in any interaction, reducing each of $A_+$ and $A_-$ by 1. To ensure sufficient cancellation to avoid overflow, we can add one clock cycle of cancellation to the end of each addition or subtraction operation.

For registers with this balanced representation, we must revisit comparison. To compare $A$ with $B$, we compare $(A_+ + B_-)$ with $(A_- + B_+)$. Since these differ by a multiple of $n^{4/5}$, our previous comparison method works. The result is that subtraction can be done with $O(1)$ additions and comparisons, which gives parallel time of $O(\log n)$.

### 6.3 Faster division by applying previous optimizations

Our most expensive operation is division by a constant, which is based on $O(\log n)$ rounds of binary search in which subtraction dominates the cost of each round. The improved cost of subtraction immediately reduces the parallel time for division to $O(\log^2 n)$ without any change to the division algorithm.

### 6.4 Converting inputs

The remaining issue is how to convert the input values in the registers, which are represented in simple unary, into the wide-digits representation. We use the unoptimized machine operations of Section 5 to create a reference value of magnitude $\Theta(n^{4/5})$ in a register and the usual base-conversion algorithms to extract the wide digits of each input register value and store them multiplied by the reference value; this initialization takes polylogarithmic parallel time, after which the per-step overhead of simulating the register machine is $O(\log^2 n)$.

Thus we have:

**Theorem 3** *The per-step bound on the number of interactions in Theorem 2 can be improved to $O(n \log^2 n)$ at the cost of an initial $O(n \log^{O(1)} n)$-interaction startup phase, where the probability of failure during either the simulation or the startup phase can be made $O(n^{-c})$ for any fixed $c$.*

## 7 Applications

### 7.1 Simulating RL

In [3], it was shown that a probabilistic population protocol with a leader could simulate a randomized LOGSPACE Turing machine with a constant number of read-only unary input tapes with polynomial slowdown. The basic technique was to use the standard reduction of Minsky [27] of a Turing machine to a counter machine, in which a Turing machine tape is first split into two stacks and then each stack is represented as a base-$b$ number stored in unary. Because the construction in [3] could only increment or decrement counters, each movement of the Turing machine head required decrementing a counter to zero in order to implement division or multiplication. Using Theorem 3, we can perform division and multiplication in $O(n \log^2 n)$ interactions, which thus gives the number of interactions for a single Turing machine step (the initial startup cost is amortized over subsequent steps for a sufficiently long execution). If we treat this quantity as $O(\log^2 n)$ parallel time, we get a simulation with polylogarithmic slowdown.

**Theorem 4** *For any fixed $c > 0$, there is a constant $d$ such that a probabilistic population protocol on a complete graph with a leader that can simulate $n^c$ steps of a randomized LOGSPACE Turing machine with a constant number of read-only unary input tapes using $d \log^2 n$ parallel time per step with a probability of failure bounded by $n^{-c}$.*

*Proof* We run a simulated register machine adjusted so that the probability of failure of each basic operations is $O(n^{-2c-1})$; since we use a constant number of register machine operations per Turing machine step, the total probability of failure in $n^c$ Turing machine steps is $O(n^c n^{-2c-1})$, which is less than $n^{-c}$ for sufficiently large $n$.

The contents of each input tape is placed in a pair of registers representing the number of 1's to the left of the read head and the number of 1's to the right of the read head. For an alphabet of size $k$, we can represent $\lfloor \log_k n/2 \rfloor$ cells of the work tape in a register holding values up to $n/2$; if we want a larger tape, we use multiple registers to hold each tape segment. The state of the finite-state controller is stored in the leader.

The head position on the work tape is represented by a pair $(i, k^s)$ where $i$ is a constant-size segment identifier stored in the leader and $k^s$ is a segment offset stored in a virtual register $S$; $(i, k^s)$ represents the position $i \lfloor \log_k (n/2) \rfloor + s$. Reading the symbol under the tape head consists of computing $(S_i / k^s) \bmod k$. Shifting the head right involves multiplying $S$ by $k$; if as a result it overflows or reaches $k^{\lfloor \log_k (n/2) \rfloor}$, we reset $S$ to 0 and increment $i$. Shifting the head left involves dividing $S$ by $k$, decrementing $i$ and resetting $S$ to $k^{\lfloor \log_k (n/2) \rfloor - 1}$ if $S$ is 1 initially. These operations require at most two divisions and some comparisons and assignments assuming the value $k^{\lfloor \log_k (n/2) \rfloor - 1}$ has been precomputed during the initialization of the simulation.

To implement random choices, the leader initializes two disjoint populations of roughly $n/2$ agents with heads or tail tokens, and chooses the result of a coin flip by waiting to see which token it sees first. This again takes at most $dn\ln^2 n$ interactions with high probability.

### 7.2 Protocols for semilinear predicates

In this section we consider the problem of computing a semilinear predicate. We first show that a simple improvement on the standard protocol of [3] reduces the expected number of interactions from $\Theta(n^2\log n)$ to $\Theta(n^2)$, even without assuming a leader. If a leader is available, Theorem 2 can be used to further reduce the number of interactions to $O(n\log^5 n)$, both in expectation and with high probability. (We do not use the faster simulation of Theorem 3 in this case because the startup time would exceed the cost of using the simpler simulation.)

The best previously known protocol for computing semilinear predicates in a probabilistic population protocol is that of [3], which proceeds in two stages: (a) a **coalescing** stage equivalent to leader election, where all agents start as candidate leaders and candidates drop out when they meet other candidates; and (b) a **broadcast** stage where the last surviving leader personally informs all other agents of the outcome of the protocol (which it computes based on data gathered during the coalescing stage). Since the expected number of interactions to eliminate one candidate given $k$ candidates is $\frac{n(n-1)}{k(k-1)}$, the coalescing stage takes

$$\begin{aligned}\sum_{k=2}^{n}\frac{n(n-1)}{k(k-1)} &= n(n-1)\sum_{k=2}^{n}\frac{1}{k(k-1)}\\ &= n(n-1)\cdot\frac{n-1}{n}\\ &= (n-1)^2\end{aligned}$$

interactions on average. But the broadcast stage is equivalent to coupon collector with a factor-of-$n$ slowdown (since interactions between non-leaders have no effect) and thus takes $\Theta(n^2\log n)$ interactions.

We can reduce the cost of the broadcast stage slightly by allowing non-leaders to recruit each other. The resulting protocol, which we call **random-walk broadcast**, has state space $\{0,1\}\times Y$ where 0 indicates a non-leader, 1 a leader, and $Y$ is the output alphabet ($\{0,1\}$ when computing a predicate). Its transitions are given by

$$(b,y),(0,y')\mapsto(b,y),(0,y)$$

with all other transitions being no-ops. The intuition is that any interaction between two non-leaders with different output values is equally likely to propagate one or the other, as each non-leader is equally likely to be the initiator. If we map the output alphabet to correct and incorrect values, the number of correct values is driven in a random walk by such interactions. But the leader cannot be persuaded by non-leaders and produces a bias in the direction of the absorbing state in which all agents have the correct output.

**Theorem 5** *Starting from any initial configuration with a single leader, the random walk broadcast protocol converges to a configuration in which all agents have the same output value in an expected $O(n^2)$ interactions.*

*Proof* The number of agents whose answers agree with a unique leader (including the leader) is a Markov chain on $1,\ldots,n$, where $n$ is the size of the population. We are interested in the maximum hitting time of the chain to $n$.

Let $T_n(k)$ be the hitting time where $k$ is the number of agents that agree with the leader. Then we have $T_n(n)=0$ and the recurrence

$$\begin{aligned}T_n(k) = \frac{n(n-1)}{2k(n-k)} &+ \frac{k(n-k)+1\cdot(n-k)}{2k(n-k)}T_n(k+1)\\ &+ \frac{k(n-k)-1\cdot(n-k)}{2k(n-k)}T_n(k-1)\end{aligned}$$

or

$$T_n(k) = \frac{n(n-1)}{2k(n-k)} + \frac{k+1}{2k}T_n(k+1) + \frac{k-1}{2k}T_n(k-1),$$

which we can rewrite as

$$\frac{2kT_n(k)}{n(n-1)} = \frac{1}{n-k} + \frac{k+1}{n(n-1)}T_n(k+1) + \frac{k-1}{n(n-1)}T_n(k-1).$$

Let $U_n(k) := \frac{kT_n(k)}{n(n-1)}$. Then

$$2U_n(k) = \frac{1}{n-k} + U_n(k+1) + U_n(k-1).$$

The solution is

$$U_n(k) = (n-k)\sum_{j=n-k+1}^{n}\frac{1}{j}.$$

Clearly, $U_n(0)=0$. We check the recurrence:

$$\begin{aligned}2U_n(k) &= 2(n-k)\sum_{j=n-k+1}^{n}\frac{1}{j}\\ &= (n-(k+1))\sum_{j=n-k+1}^{n}\frac{1}{j} + (n-(k-1))\sum_{j=n-k+1}^{n}\frac{1}{j}\\ &= U_n(k+1) - \frac{n-k-1}{n-k} + U_n(k-1) + \frac{n-k+1}{n-k+1}\\ &= \frac{1}{n-k} + U_n(k+1) + U_n(k-1).\end{aligned}$$

Thus

$$T_n(k) = \frac{n(n-1)(n-k)}{k}\sum_{j=n-k+1}^{n}\frac{1}{j}$$

and $T_n(k)\le T_n(1) = (n-1)^2$.

Now let us consider what we can do with a leader. From [3] we have that it is sufficient to be able to compute congruence modulo $k$, $+$, and $<$ to compute any semilinear predicate. From Theorem 2 we have that all of these operations can be computed with a leader in $O(n\log^5 n)$ interactions with high probability. The final stage of broadcasting the result to all agents can also be performed in $O(n\log n)$ interactions with high probability using an epidemic.

However, there is some chance of never converging to the correct answer if the protocol fails. To eliminate this possibility, we construct an optimistic hybrid protocol in which the fast but potentially inaccurate $O(n\log^5 n)$-interaction protocol is supplemented by an $O(n^2)$ leaderless protocol, with the leader choosing (in case of disagreement) to switch its output from that of the fast protocol to that of the slow protocol when it is likely the slow protocol has finished. The resulting hybrid protocol converges to the correct answer in all executions while still converging in $O(n\log^5 n)$ interactions in expectation and with high probability.

**Theorem 6** *For any semilinear predicate P, and for any $c > 0$, there is a probabilistic population protocol on a complete graph with a leader to compute P without error that converges in $O(n\log^5 n)$ interactions with probability at least $1-n^{-c}$ and in expectation.*

*Proof* First apply Theorem 6 to evaluate $P$ and broadcast the result in $O(n\log^5 n)$ interactions with probability of error at most $1-n^{-2c-7}$. To eliminate the error, we will in parallel run the $O(n^2)$ coalescing protocol of [3] as modified to use random-walk broadcast. The output of the protocol will switch from the fast algorithm to the slow one only when the second has converged with high probability; if the fast algorithm is correct, this has no effect on the output and does not increase the convergence time. But if the fast algorithm is incorrect, the slow algorithm saves it, by having the leader personally write the slow algorithm output on each agent.

To simplify the analysis of the coalescing algorithm, we assume that the leader retains its candidate bit even when interacting with another candidate. This enforces that the final remaining candidate will in fact be the leader, and that its output value will converge to the correct value. Since no candidate survives an encounter with the leader, once the leader meets every other agent it has the correct output value in the coalescing protocol. This is an instance of coupon collector—slowed down by a factor of $n$ on average—and so a simple application of Lemma 1 together with Chernoff bounds applied to the number of leader interactions shows that there is a constant $d$ such that the leader obtains the correct output after $dn^2\ln n < n^3$ interactions with probability at least $1-n^{-4c}$.

We now show how to switch from the possibly erroneous result (or results) of the epidemic-based protocol to the result of the coalescing protocol. Upon its first interaction as initiator, the leader recruits a single marker agent which it uses to implement a probabilistic trigger following a technique suggested in [2]: the trigger fires if the leader responds to the marker agent $2c+5$ interactions in a row without any intervening interactions.

The trigger fires on any particular leader interaction with probability at most $n^{-2c-5}$ and on any interaction with probability at most $n^{-2c-6}$. The expected number of firings in the first $n^3$ interactions is thus at most $n^{-2c-3}$ by linearity of expectation.

We now consider several possible cases, depending on whether the fast algorithm delivers the correct output to all agents within $O(n\log^5 n)$ interactions and whether the trigger fires before $n^3$ interactions.

1. Fast algorithm works, trigger fires at $n^3$ interactions or later. Here every agent has the correct output after $O(n\log^5 n)$ interactions, and they continue to have the correct output thereafter unless the slow algorithm has not converged, which occurs with probability at most $n^{-4c}$. Even conditioning on failure to converge after $n^3$ interactions, the slow algorithm runs at most an expected $O(n^2\log n)$ additional interaction before converging, contributing $O(n^3n^{-4c}) = o(1)$ to the total expected interactions.
2. Fast algorithm works, trigger fires before $n^3$ interactions. Here an unconverged slow algorithm may produce bad outputs after $O(n\log^5 n)$ interactions. This case occurs with probability at most $n^{-2c-3}$. Since the slow algorithm eventually converges in $O(n^3)$ expected interactions, this case also adds at most $n^3n^{-2c-3} = o(1)$ to the total expected interactions.
3. Fast algorithm fails. This case occurs with probability at most $n^{-2c-7}$. Here the expected number of interactions is dominated by the waiting time for the trigger, which is $O(n^{2c+6})$ interactions; this contributes $n^{2c+6}n^{-2c-7} = o(1)$ to the expectation.

Summing the error probabilities gives $n^{-4c}+n^{-2c-3}+n^{-2c-7} \ll n^{-c}$ for sufficiently large $n$. Summing the contributions of each case to the expectation gives $O(n\log^5 n)+o(1) = O(n\log^5 n)$ expected total interactions.

## 8 Removing the initial leader?

All of the results so far assume a unique initial leader agent. It still remains open whether we can build fast population protocols starting from a *uniform* initial population. In principle, we can execute an initial leader election phase using the simple algorithm of [3] (where each would-be leader drops out upon encountering another leader), then have the last surviving leader personally shut off all other agents one at a time in $O(n^2\log n)$ interactions, and restart them in the same number of interactions; however, the leader may have to wait an additional large polynomial time to be confident that it has in fact reached all agents. Even if we are willing to amortize the cost of the initial leader election phase over a long computation, a faster, proven algorithm is still desirable.

We present a construction based on an improved phase clock, which combines the original phase clock with a multi-valued generalization of the approximate majority protocol from [7]. This consensus phase clock appears to be robust, but must be started with at most a polynomial number of leaders. We also describe a "disposable" phase clock that appears to be able to generate an $O(n^\varepsilon)$-sized junta of leaders and start the consensus phase clock from an initial configuration with all agents in the same state. The combination of these two protocols appears to give a robust phase clock without requiring designated leaders in the initial configuration.[3]

These mechanisms have not been proven correct; instead, we rely on simulation results that show that they appear to work with high probability. A full proof of correctness is likely to require the development of better analytical tools.

### 8.1 Fast leader elimination

Simulation results suggest that the following protocol can start up a phase clock with high probability in $\Theta(\log n)$ parallel time. This protocol is one-way; we use $\delta(q_1,q_2)$ to denote the new state of a responder in state $q_2$ that interacts with an initiator in state $q_1$.

This protocol is the combination of several components, where transitions in each component may depend on the states of previous components. The first component allows us to make approximate coin tosses. The states are $Q_{\text{coin}} = \{0,1\}$, with initial value $x_{\text{coin}} = 0$, and the transition function is $\delta_{\text{coin}}(q,q') = 1-\pi_{\text{coin}}(q)$. Starting from any configuration, this protocol rapidly converges towards an equal proportion of agents in each state. The second component counts the number of consecutive coin values equal to 1 the agent has seen immediately prior, up to a maximum of $\ell > 0$. The states are $Q_{\text{count}} = \{0,1,\ldots,\ell\}$, $x_{\text{count}} = 0$, and the transition function $\delta_{\text{count}}(q,q') = \pi_{\text{coin}}(q)(\min\{\pi_{\text{count}}(q')+1,\ell\})$.

The third component approximates an exponential decay process. There is a parameter $k_1 \le \ell$. The states are $Q_{\text{decay1}} = \{0,1\}$, $x_{\text{decay1}} = 1$, and the update function is $\delta_{\text{decay1}}(q,q') = [\pi_{\text{count}}(q) < k_1]\pi_{\text{decay1}}(q')$. The idea is that for all $0 < \alpha < 1$ and $0 < c$, we can find $k_1$ such that with high probability, there is a period of $cn\log n$ steps where the number of agents with decay1 value of 1 is between 1 and $n^\alpha$. In this period, using agents with decay1 value of 1 as temporary leaders, we can run a *disposable* phase clock that functions correctly only for a constant number of phases before all the values of decay1 become 0. This phase clock is used to choose a stable leader population of size $\Theta(n^{1-\varepsilon})$, which in turn supports a second copy of the phase clock that runs for polynomially many steps.

[3] Much of the material in this section originally appeared in [7].

### 8.2 A consensus-enforcing phase clock

Our consensus variant of the phase clock works as follows. In addition to the phases $0,1,\ldots,\phi-1$, we have a blank "phase" $b$. Thus $Q_{\text{phase1}} = \{b,0,1,\ldots,\phi-1\}$ and $x_{\text{phase1}} = 0$. If $x$ is a nonblank phase, then let $\text{succ}(x) = (x+1) \bmod \phi$ be the successor phase of $x$. We have

$$\delta_{\text{phase1}}(q,q') = \begin{cases} p' & \text{if } p = b \\ p & \text{if } p' = b \\ p' & \text{if } p' = p \neq b \text{ and } \pi_{\text{decay1}}(q) = 0 \\ \text{succ}(p') & \text{if } p' = p \neq b \text{ and } \pi_{\text{decay1}}(q) = 1 \\ p' & \text{if } p,p' \neq b \text{ and } p' = \text{succ}(p) \\ p & \text{if } p,p' \neq b \text{ and } \text{succ}(p') = p \\ b & \text{otherwise,} \end{cases}$$

where $p = \pi_{\text{phase1}}(q)$ and $p' = \pi_{\text{phase1}}(q')$. If the phase of the initiator is blank or one behind the responder's, the responder's phase is unchanged. If the phase of the responder is blank, it copies the phase of the initiator. If the phases are non-blank and equal, the responder increments its phase if and only if the initiator has decay1 value 1 (temporary leader status.) If the initiator's phase is one more than the responder's, the responder increments its phase. In all other cases, the responder sets its phase to blank. In summary, we are following a multiple-valued generalization of the 3-state majority algorithm *except* when the phases are nonblank and within distance 1 of one another. In this case, we revert to behavior like that of the original phase clock.

### 8.3 Combining the mechanisms

Once the disposable phase clock is running, it is used to select the real phase clock's leaders. This is accomplished by having another exponential decay process that is reset by the disposable phase clock each complete cycle. Thus we need a way to detect approximately the onset of each cycle. Our criterion is for each agent to keep a local "maximum" of the phases it has been in, and perform the reset when this maximum wraps around. Formally, $Q_{\max} = \{0,1,\ldots,\phi-1\}$, $x_{\max} = 0$, and

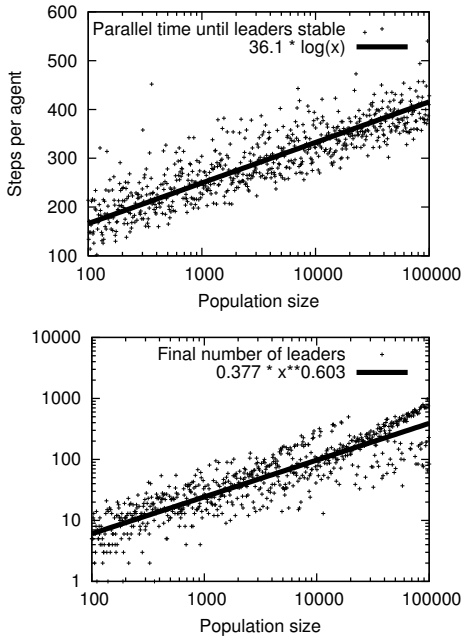
$$\delta_{\max}(q,q') = \begin{cases} p' & \text{if } p' \neq b \text{ and} \\ & \quad (p' - \pi_{\max}(q')) \bmod \phi \le \phi/2 \\ \pi_{\max}(q') & \text{otherwise,} \end{cases}$$

where $p' = \delta_{\text{phase1}}(q,q')$. Since the last cycle may be partial, we also need a one-value history for the decay process. Now $Q_{\text{decay2}} = \{0,1\}\times\{0,1\}$, $x_{\text{decay2}} = (1,1)$, and

$$\delta_{\text{decay2}}(q,q') = \begin{cases} (1,y) & \text{if } \delta_{\max}(q,q') \\ & \quad < \pi_{\max}(q') \\ ([\pi_{\text{count}}(q) < k_2]y, y') & \text{otherwise,} \end{cases}$$

where $(y,y') = \delta_{\text{decay2}}(q')$. The final set of leaders are those agents with $y' = 1$ when the disposable phase clock stops

**Fig. 1** Simulation results: parallel time of leader election and final number of leaders




running. A second copy of the consensus phase clock, running from the initial configuration using $y' = 1$ to designate leaders, rapidly converges in simulation to correct robust phase clock behavior when the number of leaders becomes appropriate.

### 8.4 Simulation results

We implemented the disposable phase clock leader election protocol and tried it once on each value of $\lfloor 1.01^n \rfloor$ between 100 and 100000 for integers $n$, with every agent in the same initial state. There are three parameters to tune: $\phi$, $k_1$, and $k_2$. The protocol is not very sensitive to the settings of these parameters, but the setting $\phi = 9$, $k_1 = 5$, and $k_2 = 4$ worked better than many others.

The results are depicted in Figure 1. As can be seen, it seems that the protocol generally leaves $\Theta(n^{1-\varepsilon})$ leaders and completely converges in $O(\log n)$ parallel time.

### 8.5 Reduction to a single leader

The distinction between one leader and a polynomial number of leaders is not important for the phase clock, but may become so in the register machine simulation; for example, a single leader can easily initialize a unary register to 1 by recruiting a spare agent, but multiple leaders will have a hard time recruiting exactly one such agent. Fortunately, once the phase clock is running, the leadership cadre can be reduced to a single leader by flipping coins. In each round, every leader chooses a 0 or 1 value and propagates it by epidemic. If both values appear, any leader with a 1 drops out and the 0 leader (or leaders) restart the register machine simulation. We thus obtain a single leader an expected $O(n \log^2 n)$ interactions after the phase clock stabilizes.

## 9 Open problems

The most pressing open problem is whether the assumption of an initial leader can be provably eliminated without drastically raising the cost of our protocols. Though the simulation results of Section 8.4 are promising, at present we have no fast, provably correct protocol for rapidly producing a single leader.

A related question is whether it is possible to make the register machine simulation and other constructions robust against errors. The redundancy inherent in the wide-digit representation of Section 6.1 together with a junta of leaders running in lockstep suggests the possibility of tolerating crash failures where agents drop out of the population, though detailed analysis will be needed to ensure that errors do not accumulate over long computations. Byzantine failures seem much more dangerous, as a single Byzantine agent could initiate new epidemics that quickly overwhelm and disrupt the correct computation. Here the possibility of "dueling epidemics" to constrain the effects of Byzantine agents, as in the Byzantine-resistant approximate majority protocol of [7], or the adoption of an extended model incorporating minimal identity information, as used in the Byzantine-resistant *community protocols* of Guerraoui and Ruppert [24], may give reason for hope.

Finally, it would be interesting to explore the effect of assuming a interaction distribution that is non-uniform—perhaps even one that changes over time—to reflect the physical effects of spatial dispersion and/or movement of the agents.

## 10 Acknowledgments

Part of this work was done while the third author was a student at the University of Rochester and at Princeton University.

## References

1. Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In Viktor K. Prasanna, Sitharama Iyengar, Paul Spirakis, and Matt Welsh, editors, *Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USA, June/July, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer-Verlag, June 2005.
2. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Urn automata. Technical Report YALEU/DCS/TR-1280, Yale University Department of Computer Science, November 2003.

3. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC '04: Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, pages 290–299. ACM Press, 2004.
4. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, March 2006.
5. Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. In *Distributed Computing: 20th International Symposium, DISC 2006: Stockholm, Sweden, September 2006: Proceedings*, pages 61–75, September 2006.
6. Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, pages 292–299, July 2006.
7. Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. In Andrzej Pelc, editor, *Distributed Computing, 21st International Symposium, DISC 2007, Lemesos, Cyprus, September 24-26, 2007, Proceedings*, volume 4731 of *Lecture Notes in Computer Science*, pages 20–32. Springer, September 2007.
8. Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. Unpublished manuscript submitted to *Distributed Computing*., January 2008.
9. Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. On the power of anonymous one-way communication. In *Ninth International Conference on Principles of Distributed Systems*, pages 307–318, December 2005.
10. Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.
11. Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. In *Ninth International Conference on Principles of Distributed Systems*, pages 79–90, December 2005.
12. Dana Angluin, Michael J. Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *Proceedings of the Second IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '06)*, pages 37–50, 2006.
13. Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. In Sam Toueg, Paul G. Spirakis, and Lefteris M. Kirousis, editors, *Distributed Algorithms, 5th International Workshop*, volume 579 of *Lecture Notes in Computer Science*, pages 71–79, Delphi, Greece, 1991. Springer-Verlag.
14. Norman T. J. Bailey. *The Mathematical Theory of Infectious Diseases, Second Edition*. Charles Griffin & Co., London and High Wycombe, 1975.
15. Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.
16. D. J. Daley and D. G. Kendall. Stochastic rumours. *Journal of the Institute of Mathematics and its Applications*, 1:42–55, 1965.
17. Ariel Daliot, Danny Dolev, and Hanna Parnas. Self-stabilizing pulse synchronization inspired by biological pacemaker networks. In Shing-Tsaan Huang and Ted Herman, editors, *Self-Stabilizing Systems*, volume 2704 of *Lecture Notes in Computer Science*, pages 32–48. Springer, 2003.
18. Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *Proceedings of the Second IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '06)*, pages 51–66, 2006.
19. Zoë Diamadi and Michael J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1–2):72–82, March 2001. Also appears as Yale Technical Report TR–1207, January 2001.
20. Shlomi Dolev and Jennifer L. Welch. Self-stabilizing clock synchronization in the presence of Byzantine faults. *Journal of the ACM*, 51(5):780–799, 2004.
21. Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104:1876–1880, 2000.
22. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
23. Daniel T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.
24. Rachid Guerraoui and Eric Ruppert. Even small birds are unique: Population protocols with identifiers. Technical Report CSE-2007-04, Department of Computer Science and Engineering, York University, 2007.
25. Ted Herman. Phase clocks for transient fault repair. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1048–1057, 2000.
26. A. P. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Structures and Algorithms*, 7:59–80, 1995.
27. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.
28. Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes-Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101, Warszawa, 1929.