Yale University
Department of Computer Science

Fast Computation by Population Protocols With a Leader

Dana Angluin James Aspnes David Eisenstat

YALEU/DCS/TR-1358
May 2006

Fast Computation by Population Protocols With a Leader

Dana Angluin* James Aspnes*† David Eisenstat‡

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 in which standard arithmetic operations like comparison, addition, subtraction, multiplication, and division can be simulated in $O(n \log^4 n)$ interactions with high probability. Applications include a reduction of the cost of computing a semilinear predicate to $O(n \log^4 n)$ interactions from the previously best-known bound of $O(n^2 \log n)$ interactions and simulation of a LOGSPACE Turing machine using the same $O(n \log^4 n)$ interactions per step. These bounds on interactions translate into $O(\log^4 n)$ time per step in a natural model in which each agent participates in an expected $\Theta(1)$ interactions per time unit. 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.

1 Introduction

The **population protocol** model of Angluin *et al.* [AAD$^+$04] 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**.

In the simplest version of the model, any agent may interact with any other, 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** [Pre29]), where semilinearity was shown to be sufficient in [AAD$^+$04] and necessary in [AAE06]. 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.

One fundamental measure of convergence is the total number of interactions until all agents have the correct value, considered as a function of n, the number of agents in the population. Assuming that interactions

*Department of Computer Science, Yale University.

†Supported in part by NSF grants CNS-0305258 and CNS-0435201.

‡Department of Computer Science, University of Rochester.

may occur in parallel, we are also interested in related measures of time in which each agent participates in $\Theta(1)$ interactions per time unit and time 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 [AAD$^+$04], in which each interaction occurs between a pair of agents chosen uniformly at random. In this model, it was shown in [AAD$^+$04] that any semilinear predicate can be computed in $\Theta(n^2 \log n)$ expected interactions using a protocol based on leader election in which the leader communicates the outcome by interacting with every other agent. Allowing answers to be incorrect with inverse polynomial probability, protocols also were given to simulate randomized LOGSPACE computations, with polynomial slowdown.

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. The simulated registers support the usual arithmetic operations, including addition, subtraction, multiplication, division, and comparison, with implementations that complete with high probability in $O(n \log^4 n)$ interactions and polylogarithmic time per operation. As a consequence, any semilinear predicate can be computed without error by a probabilistic population protocol that converges in $O(n \log^4 n)$ interactions with high probability, and randomized LOGSPACE computation can be simulated with inverse polynomial error with only polylogarithmic slowdown.

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. If a leader is not provided, it is in principle possible to elect one; however, the best known expected time for leader election in a population protocol is still the $\Theta(n^2)$ interactions or $\Theta(n)$ time of a naive protocol in which candidate leaders drop out only on encountering other leaders. There must also be a way to reinitialize the simulation protocol once all but one of the candidates drops out. We discuss these issues further in Section 8.

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 time 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 11. Applications to simulating LOGSPACE Turing machines and computing semilinear predicates are described in Section 6 and Section 7. Some directions for future work are described in Section 8.

Many of our results are probabilistic, and our algorithms include timing parameters that can be used to adjust the probability of success. We say that a statement holds **with high probability** if for any constant c there is a setting of the timing parameters that cause the statement to hold with probability at least $1 - n^{-c}$.

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 [DF01],

and a related model of urn automata was explored in [AAD+03]. 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 [AAD+04, AAC+05, AAD+06], various forms of one-way and delayed communication [AAER05], and failures of agents [DGFGR06]. The properties computed by population protocols have also been extended from predicates on the initial population to predicates on the underlying interaction graph [AAC+05], self-stabilizing behaviors [AAFJ05] and stabilizing consensus [AFJ06].

Similar systems of pairwise interaction have previously been used to model the interaction of molecules in solution and the propagation in a human population of rumors or epidemics of infectious disease [Bai57, DK65, Gil92]. 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.* [BHO+99] 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 β 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. [Her00]. There is a substantial stream of research on building self-stabilizing synchronized clocks dating back to to the work of Arora *et al.* [ADG91]. Recent work such as [DW04] 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 that is available in a population protocol. An intriguing protocol of Daliot *et al.* [DDP03] 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

Because we consider only the all-pairs interaction graph, 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, 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. A **configuration** C is a function from the population to states. 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**. An **execution** is a sequence $C_1, C_2, \ldots$ of configurations such that for each i, $C_i \to C_{i+1}$. 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}. To extend our results to apply to time in a parallel model, we assume that interactions are triggered by a Poisson process with uniform rate $\Theta(1/n)$ for each possible interaction; this yields an average of $\Theta(1)$ interactions per agent per time unit and justifies the assumption that with high

probability, time is simply number of interactions divided by n.

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).

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 time for the epidemic to finish is $\Theta(n \log n)$ with high probability.

To do so, we reduce the running time of the epidemic protocol to the running time of 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 time to fill the last k of n bins based on an occupancy bound of Kamath *et al.* [KMPS95].

Lemma 1 *Let $S(k, n)$ be the time to fill the last k of n bins in the coupon collector problem. Then for any fixed $\epsilon > 0$ and $c > 0$, there exist positive constants c_1 and c_2 such that for sufficiently large n and any $k > n^{\epsilon}$, $c_1 n \ln k \leq S(k, n) \leq c_2 n \ln k$ 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 [KMPS95] states that with m balls tossed uniformly into n bins,

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

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 at some time m. Substitute $n = k$ and $m = (1/4)k \ln k$ in (3.1). 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 expected number of interactions to infect the last k agents in a one-way epidemic. We bound $E[T(k)]$ as follows.

$$E[T(k)] = \sum_{i=1,k} n(n-1)/i(n-i) = (n-1)(H(k) + H(n-1) - H(n-1-k)).$$

If $k \leq n/2$ then $(H(n-1) - H(n-1-k))$ is bounded by $\ln 2$ and in this case,

$$(n-1)H(k) \leq E[T(k)] \leq (n-1)(H(k) + \ln 2).$$

To get concentration bounds for $T(k)$, we now reduce the time for an epidemic to the time 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.

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 $\epsilon > 0$ and $c > 0$, there exist positive constants c_1 and c_2 such that for sufficiently large n and any $k > n^{\epsilon}$, $c_1 n \ln k \leq T(k) \leq c_2 n \ln k$ 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 time to infect the first $k-1$ susceptibles—is identical to that of $T(n) - T(n-k+1)$—the time to infect the last $k-1$.

Next we'll bound the time 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 time 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 time 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 time 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. 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 case where a leader in phase i responds to an initiator in phase $i+1 \bmod m$ through $i + m/2 \bmod m$ does not arise in the normal operation of the phase clock. We can treat such an occurrence as signaling an error.

A **round** consists of m phases. A new round starts when the leader enters phase 0.

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

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

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.

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 time $t + an \ln n$. 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 time $t + an \ln n$ 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 time $t + c_1 n \ln n^{1/4} = t + (c_1/4) n \ln n$ 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 time $t + an \ln n$ when $a = c_1/4$ is at most $n^{-1} + n^{-3/4} \leq n^{-1/2}$. ∎

Corollary 4 *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. ∎

Theorem 5 *For any fixed $c, d > 0$, there exists a constant m such that, for sufficiently large n, 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 $dn \ln n$ with probability at least $1 - n^{-c}$.*

Proof: The essential idea is to apply Corollary 4 twice: once to show that the time 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 time $c_2 n \ln n$ with probability at least $1 - n^{-3c}$. By Corollary 4, 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 4 there is a constant k_2 such that the number of interactions to complete k_2 consecutive phases is at least $dn \ln n$ with probability at least $1-n^{-3c}$. Setting $m \geq k_2$ thus gives that all n^c rounds take at least $dn \ln n$ interactions with probability at least $1 - n^c n^{-3c} = 1 - n^{-2c}$. Thus the total probability of failure is bounded by $2mn^{-2c} + n^{-2c} < n^{-c}$ for sufficiently large n as claimed. ∎

3.3 Duplication

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

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

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 6 *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 7 *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 $\Pr[A_t] \leq \left(1 - \frac{1}{4n}\right)^t$. The base case is trivial. For the induction step, we have $\Pr[A_t] = \Pr[A_t|A_{t-1}]\Pr[A_{t-1}] + \Pr[A_t|A_{t-1} = 0]\Pr[A_{t-1} = 0] = \Pr[A_t|A_{t-1}]\Pr[A_{t-1}] \leq \Pr[A_t|A_{t-1}]\left(1 - \frac{1}{4n}\right)^{t-1}$.

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 $\Pr[A_t] \leq \Pr[A_t|A_{t-1}]\left(1 - \frac{1}{4n}\right)^{t-1} \leq \left(1 - \frac{1}{4n}\right)^t$.

Setting $t = 4(c+1)n \ln n$ gives $\Pr[A_t] \leq (1 - \frac{1}{4n})^{4(c+1)n \ln n} \leq \exp(-\frac{4(c+1)n \ln n}{4n}) = n^{-c-1}$. 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 exists. It uses three states (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

$$(0, y) \mapsto (0, y)$$
$$(1, y) \mapsto (1, 2)$$

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 8 *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) there is no agent that satisfies the predicate and every agent is in state 1, or (b) there is some agent that 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 $\Theta(\log n)$ time.

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.

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 a bit A_i and the value of A is simply $\sum_i A_i$.

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. We omit describing the underlying transitions as the details are tedious.

Observe that 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 time is 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 8). 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 6), and the CANCEL operation either converges or leaves at most $n/4$ uncancelled values (Lemma 7). Note that for either operation, which outcome occurred can be detected with COPY and PROBE operations.

If the leader wishes these operations to succeed, it must wait for $\Omega(n \log n)$ interactions, where the constant is chosen based on the desired error bound. But this can be done using a phase clock with appropriate parameter (Theorem 5): if it is large enough that both the probability that an operation completes too late and the probability that some phase clock triggers to early is $o(n^{-2c})$ per operation, then the total probability that one of n^c operations fails is $o(n^{-c})$.

Operation	Effect	Implementation	Notes
Constant 0	$A \leftarrow 0$	SET($\neg A$)	
Constant 1	$A \leftarrow 1$	SET($\neg A$) $A_{\text{leader}} \leftarrow 1$	
Assignment	$A \leftarrow B$	COPY(B, A)	
Addition	$A \leftarrow A + B$	COPY(B, X) DUP(X, A) 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.

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

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 7, 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. ∎

Algorithm 1 Comparison algorithm.

$A' \leftarrow A$.
$B' \leftarrow B$.
$C \leftarrow 1$.
$r \leftarrow 0$.
while true **do**
  CANCEL(A', B').
  if $A' = 0$ and $B' = 0$ **then**
    return $A = B$.
  else if $A' = 0$ **then**
    return $A < B$.
  else if $B' = 0$ **then**
    return $A > B$.
  end if
  $r \leftarrow 1 - r$.
  if $r = 0$ **then**
    $C \leftarrow C + C$.
    if addition failed **then**
      return $A = B$.
    end if
  end if
  $A' \leftarrow A' + A'$.
  $B' \leftarrow B' + B'$.
end while

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. 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.

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

Proof: By Lemma 7, 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 $A \bmod k$. Since each iteration adds one bit to the quotient, there are at most $O(\lg n)$ iterations of the outer loop, for a total cost of $O(\lg^4 n)$ microcode operations (since each outer loop iteration requires one subtraction operation).

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 the results of the preceding sections gives:

Theorem 11 *A probabilistic population 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* $O(n \log^4 n)$ *interactions can be made* $O(n^{-c})$ *for any fixed* c*.*

6 Simulating RL

In [AAD+04], 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 [Min67] 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 [AAD+04] 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 11, we can perform division and multiplication in $O(n \log^4 n)$ interactions, which thus gives the number of interactions for a single Turing machine step. If we treat this quantity as $O(\log^4 n)$ time, we get a simulation with polylogarithmic slowdown.

Theorem 12 *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 $d \log^4 n$ 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 $O(n^c n^{-2c-1}) < n^{-c}$ 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 comparisions 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 $dn \ln^4 n$ interactions with high probability. ∎

7 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 [AAD+04] 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 11 can be used to further reduce the number of interactions to $O(n \log^4 n)$, both in expectation and with high probability.

The best previously known protocol for computing semilinear predicates in a probabilistic population protocol is that of [AAD+04], 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 waiting time to eliminate one candidate given k candidates is $\frac{n(n-1)}{k(k-1)}$, the coalescing stage takes

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

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 13 *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

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

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 $T_n(k) \leq T_n(1) = (n-1)^2$. ∎

Now let us consider what we can do with a leader. From [AAD$^+$04] we have that it is sufficient to be able to compute congruence $\bmod k$, $+$, and $<$ to compute any semilinear predicate. From Theorem 11 we have that all of these operations can be computed with a leader in $O(n \log^4 n)$ 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 $O(n \log^4 n)$-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 $O(n \log^4 n)$ interactions in expectation and with high probability.

Theorem 14 *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 $O(n \log^4 n)$ interactions with probability at least $1 - n^{-c}$ and in expectation.*

Proof: First apply Theorem 14 to evaluate P and broadcast the result in $O(n \log^4 n)$ 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 [AAD$^+$04] 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 [AAD$^+$03]: 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 $O(n \log^4 n)$ interactions and whether the trigger fires before n^3 interactions.

1. Fast algorithms works, trigger fires at n^3 interactions or later. Here every agent has the correct output after $O(n \log^4 n)$ 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 $O(n^3 n^{-4c}) = o(1)$ 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 $O(n \log^4 n)$ time. 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 $n^3 n^{-2c-3} = o(1)$ to the total expected interactions.

3. Fast algorithm fails. This case occurs with probability at most n^{-2c-7}. Here the expected time 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 $O(n \log^4 n) + o(1) = O(n \log^4 n)$ expected total interactions. ∎

8 Open problems

For most of the paper, we have assumed that a unique leader agent is provided in the initial input. The most pressing open problem is whether this assumption can be eliminated without drastically raising the cost of our protocols.

One problem is the question of whether we can efficiently restart the phase clock after completing an initial leader election phase. A proof of possibility can be obtained by observing that the leader can shut off all other agents one at a time in $O(n^2 \log n)$ interaction, and then restart them in the same time; however, the leader may have to wait an additional large polynomial time to be confident that it has in fact reached all agents. We believe, based on preliminary simulation results, that a modified version of our phase clock can

be restarted much more efficiently by a newly-elected leader. This would allow us to use our LOGSPACE simulator after an initial $O(n^2)$-interaction leader election stage. But more work is still needed.

Even better would be a phase clock that required no leader at all. This would allow every agent to independently simulate the single leader, eliminating both any initial leader election stage and the need to disseminate instructions. Whether such a leaderless phase clock is possible is not clear.

References

[AAC+05] 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.

[AAD+03] 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.

[AAD+04] 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.

[AAD+06] 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.

[AAE06] Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. To appear, PODC 2006, July 2006.

[AAER05] 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.

[AAFJ05] 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.

[ADG91] 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.

[AFJ06] Dana Angluin, Michael J. Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. To appear in Proc. International Conference on Distributed Computing in Sensor Systems (DCOSS06), June 2006.

[Bai57] Norman T. J. Bailey. *The Mathematical Theory of Epidemics*. Charles Griffin & Co., London, 1957.

[BHO+99] 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.

[DDP03] 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.

[DF01] 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.

[DGFGR06] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. To appear in Proc. International Conference on Distributed Computing in Sensor Systems (DCOSS06), June 2006.

[DK65] D. J. Daley and D. G. Kendall. Stochastic rumours. *Journal of the Institute of Mathematics and its Applications*, 1:42–55, 1965.

[DW04] 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.

[Gil92] Daniel T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.

[Her00] Ted Herman. Phase clocks for transient fault repair. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1048–1057, 2000.

[KMPS95] 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.

[Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[Pre29] 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.