THE DISTRIBUTED COMPUTING COLUMN

BY

STEFAN SCHMID

Faculty of Computer Science, University of Vienna
Währinger Strasse 29, AT - 1090 Vienna
stefan_schmid@univie.ac.at

In this issue of the distributed computing column, Robert Elsässer and Tomasz Radzik provide a review of interesting recent models and results for population protocols, with a focus on exact majority and leader election.

Many thanks to the authors for their contribution and enjoy!

PS: For those dear readers interested in learning even more about population protocols, I would like to point out that it happens that the September 2018 issue of the SIGACT News column also features an overview article on population protocols, by Alistarh and Gelashvili, revolving around models where agents in the system can have more than a constant amount of local memory.

Recent Results in Population Protocols for Exact Majority and Leader Election

Robert Elsässer
Department of Computer Sciences
University of Salzburg
elsa@cs.sbg.ac.at

Tomasz Radzik
Department of Infomatics
King's College London
tomasz.radzik@kcl.ac.uk

Abstract

Population protocols act in a simple and natural framework to solve fundamental problems in networks. Given a population of n anonymous nodes (agents), a scheduler chooses in discrete time steps two nodes for interaction, which then exchange their current states and perform a so called state transition. We focus on the random scheduler, which selects in each step two nodes uniformly at random for interaction, and on the problems of exact majority and leader election. In exact majority, initially the nodes possess one of two opinions, A or B, and the population is required to converge to a configuration, in which every node has the opinion of the initial majority. In leader election, each node starts with the same state, and the system should converge to a configuration, with exactly one node in a leader state. The goal is to design protocols, which require a small number of states and reach a correct final configuration as fast as possible. In this paper, we give an overview of the population protocols for these problems, focusing on the most recent results.

1 Introduction

Population protocols, introduced by Angluin *et al.* [5], act in a simple and natural computational framework to solve certain fundamental problems in networks. In the underlying communication model, the system consists of n anonymous nodes (also referred to as agents or individuals) and a scheduler which selects in discrete time steps pairs of nodes for interaction. Two interacting nodes exchange their current states and perform a *state transition*, which is specified in the population protocol. The system is expected to stabilize eventually in configurations which obey the *output property* specified in the problem definition. For example, in the leader election problem, all nodes start in the same state and the system should eventually stabilize in a configuration with exactly one node outputting a leader state and all the other nodes in follower states.

The population is modeled by a graph, and the nodes are connected according to the edges of this graph. Interactions are only allowed between adjacent nodes, that is, the scheduler can only choose pairs of nodes that are connected. A protocol is specified by a *state transition function* (or more generally, by a *transition algorithm*), which is applied at each interaction. The complexity measures of a protocol are the size of the state space of the transition function and the convergence time to the output property. Thus given a problem and a graph modeling a population, the goal is to design a protocol which has a small state space and converges fast to a correct configuration.

Angluin *et al.* [5] motivated the population protocols model by simple computations in sensor networks. They gave an example of a flock of birds, where each bird is equipped with a simple limited-capacity sensor. Each sensor becomes activated upon receiving a global start signal, and two sensors can only communicate if they are close enough to each other. The sensors are able to measure the temperature of the birds and the goal is to recognize whether a certain threshold number of birds in the flock have elevated temperature. Other examples motivating population protocols come from chemical reaction networks [42] and from relations between population protocols and Petri nets and vector addition systems [31]. Chen *et al.* [23] showed that population protocols can be implemented at the level of DNA molecules, and Cardelli and Csiksz-Nagy [22] discussed similarities of structure and dynamics between some biochemical regulatory processes in living cells and some population protocols.

In the original definition of population protocols, the actions of each node are goverened by the same finite state machine, which describes the state transitions. The finite state space is independent of the size of the population, meaning that the nodes, as computational devices, possess only a constant number of bits. All nodes follow the same protocol, i.e., two interacting nodes perform the state transition defined by the finite state machine applied to the current states of the nodes. The nodes are indistinguishable, i.e., there are no node identifiers. The scheduler is usually governed by adversary, but the schedule (the infinite sequence of pairs of nodes chosen for interaction) must fulfill some global minimum fairness condition to ensure progress of computation. Finally, while we expect that the system converges to the correct output, we cannot require that the nodes detect when a stable configuration is reached. See Aspnes and Ruppert [11] for a comprehensive discussion of various aspects of the population protocols model.

When the population protocols were introduced, the main question was to characterize the class of problems which are solvable in this model. In the basic model population protocols compute exactly predicates definable in Presburger arithmetic [5, 6, 9]. The question of computational power of population protocols has been considered also under different variants of the communication model or for certain classes of interaction graphs, such as bounded-degree graphs [5, 21]. In the case of an adversarial scheduler, the worst case interaction graph is the clique, as any schedule on an arbitrary graph can be simulated by a sequence of interactions on a complete graph.

If the scheduler is governed by an adversary, then it is impossible to reason about the convergence time of population protocols. Therefore, in most cases when the purpose of study is to analyze and compare convergence times of population protocols, a random scheduler is assumed. This means that at each discrete time step, two nodes are selected uniformly at random for interaction. In recent years, to study trade-offs between memory and time, the constant memory restriction has been relaxed and population protocols with the state space growing with the number of nodes have been analyzed.

1.1 Model

A population protocol and its time performance can be formalized as follows (see e.g. [15]). Let V denote the population, that is, the set of nodes; $n = |V|$. All protocols which we consider in this paper assume the complete graph of node connections, that is, any pair of nodes can interact. Let S be the set of states of the protocol, which can have constant size or its size may grow with n. Let $q(v,t) \in S$ denote the state of a node $v \in V$ at step t (i.e., after t individual interactions), where $q(v,0)$ is the initial state of v. Two interacting nodes perform a state transition according to a common deterministic function $\delta : S \times S \rightarrow S \times S$. Transition $\delta(q', q'') = (r', r'')$ means that if a node v' in state q' interacts with a node v'' in state q'', then v' changes its state to r' and v'' to r''. A population protocol has also an *output function* $\gamma : S \rightarrow \Gamma$, which is referred to in the specification of the desired output property of the computation. Here Γ is the set of output values.

If interactions between nodes are undirected (or symmetric), then transition $\delta(q', q'') = (r', r'')$ implies that $\delta(q'', q') = (r'', r')$. In particular, for each state q, there is a transition $\delta(q,q) = (r,r)$ for some state r. If interactions are directed (asymmetric), then one of the two interacting nodes is the caller (or initiator) and the other one is the callee (or responder). In this case we can have symmetry breaking transitions $\delta(q,q) = (r', r'')$, where $r' \neq r''$. In this paper we consider undirected interactions, unless stated otherwise.

We say that the system is in a *stable configuration*, if no sequence of interactions can change the outputs of the nodes. The computation continues (it is perpetual) and nodes may continue updating their states, but if a node changes from a state q to another state q', then the outputs $\gamma(q')$ and $\gamma(q)$ are the same. The main types of output guarantees are *always correct* and *w.h.p.*[1] *correct*. An always correct protocol reaches a correct stable configuration with probability 1. A *w.h.p.*-correct protocol reaches the correct stable configuration with high probability, but with some low but positive probability it may stabilize in incorrect configurations or may not stabilize at all.

The *sequential time* is defined as the number of interactions. The *parallel time* counts the number of *rounds*, each consisting of n interactions. In this paper we assume the random scheduler, so each node participates on average in 2 interactions in one round, and we always give the parallel time of a protocol.

The most frequently used notions of time performance of a population protocol are the *stabilization time* and the *convergence time*. The stabilization time T_S is defined as the first round when the system enters a correct stable configuration. The convergence time T_C measures the number of rounds until the system has reached and remains in configurations, which have the correct output property required in the problem specification. In the case of a random scheduler T_S and T_C are random variables. We are interested in population protocols for which the stabilization time T_S, or convergence time T_C, is small *w.h.p.* and the expectation $\mathbb{E}(T_S)$, resp. $\mathbb{E}(T_C)$, is finite and ideally also small. A finite expectation implies that the protocol is always correct. Note that $T_C \le T_S$ since a stable configuration implies that the system will output the same values in all subsequent steps.

A protocol is *uniform*, if it does not refer to the number of nodes or its estimate, that is, it uses the same transition function for any population size. Population protocols with constant number of states are uniform, but protocols with the state space growing with the increasing population tend to be nonuniform. For example, a nonuniform protocol may specify that a node repeats some operation for $\log n$ interactions. An example of a uniform protocol with growing state space is the asymmetric protocol with transitions $\delta(q_i, q_i) = (q_i, q_{i+1})$ and $\delta(q_i, q_j) = (q_i, q_j)$, for $i, j \ge 1$ and $i \ne j$. Recent papers which focus on designing uniform population protocols include [15, 26, 27]. A formal model of uniform protocols is given in [26, 27] and assumes that transition functions are specified by Turing Machines. The same Turing Machine (TM) is used for populations of any size. Each node has a copy of this TM and the current state of this node is the content of the tape(s) of its TM.

[1]A property $P(n)$, *e.g.* that a given protocol reaches a stable correct configuration, holds *w.h.p.* (with high probability), if it holds with probability at least $1 - n^{-\alpha}$, where constant $\alpha > 0$ can be made arbitrarily large by changing the constant parameters in $P(n)$ (*e.g.* the constant parameters of a protocol).

1.2 Exact majority and leader election

Probably the most well-studied problems in this area are exact majority and leader election. In the following, we will define these problems separately, and in Sections 4 and 5 we are going to discuss the most recent results and their analysis in more detail.

Exact majority. At the beginning, each node has opinion A or B and we assume that one of these two opinions is in majority. Let a_0 and b_0 denote the initial number of nodes with opinion A and B, respectively. The goal is to design a population protocol such that the system eventually stabilizes in a configuration in which all nodes know which opinion has larger support. That is, if $a_0 > b_0$, then all nodes eventually output A, if $a_0 < b_0$, then all nodes eventually output B, and there is no output requirement, if $a_0 = b_0$. Formally, we assume that initially each node is in one of two states q_A and q_B, and if, say, $a_0 > b_0$, for any fair sequence of interactions Υ, there is some t_Υ such that for any $t > t_\Upsilon$, we have $\gamma(q(v,t)) = A$. In this paper we are mainly interested in *exact* majority, meaning we want protocols, which work whenever the initial imbalance between the opinions $|a_0 - b_0|$ is positive, even if it is just 1. In contrast, *approximate-majority* protocols may not work, if the initial imbalance is below some threshold. When we simply refer to "majority", we mean exact majority.

Leader election. Initially all nodes are in the same state and the goal is that eventually exactly one node is in a leader state, while all other nodes are in follower states. Formally, the set of output values is $\Gamma = \{L, F\}$ and for any fair sequence of interactions Υ, there is some t_Υ such that for any $t > t_\Upsilon$, we have $\gamma(q(v,t)) = L$ for exactly one node v, and $\gamma(q(u,t)) = F$ for all other nodes u.

For both problems the objective is to design population protocols which solve the problem and require a small number of states and rounds. On one side, there are different types of lower bound results, which bound the number of states given certain time requirements, or bound the time given certain requirements on the number of states. On the other, more extensive side, a number of exact-majority and leader election protocols exist with specific upper bounds on both the number of states and (stabilization and/or convergence) time.

In the next section we give an overview of protocols for these two problems, focusing on recent results. In Section 3 we give some basic probabilistic results, which are often used in the protocols we consider. This is followed by more detailed discussion of exact-majority protocols in Section 4 and leader election protocols in Section 5. For exact majority (Section 4), we first explain the canceling-doubling framework, which was introduced by Angluin *et al.* [7], and then discuss how this framework was implemented in protocols proposed by Angluin *et al.* [7], Bilke *et al.* [20], Alistarh *et al.* [2] and Berenbrink *et al.* [15]. The protocol proposed in [15], which has $O(\log n)$ states and $O(\log^{5/3} n)$ stabilization time *w.h.p.*

and in expectation, is currently the fastest population protocol for exact majority with $O(\text{polylog}\, n)$ states. The descriptions of the canceling-doubling framework in Section 4.1 and of the algorithms in Sections 4.3-4.5 are (almost) the same as in [15] w.r.t. style, wording and formulation.

For leader election (Section 5), we outline the protocols proposed by Alistarh *et al.* [1], Bilke *et al.* [20], Alistarh *et al.* [2], Berenbrink et al. [19] and Gąsieniec and Stachowiak [33]. The protocol proposed in [33] is the first $O(\text{polylog}\, n)$-stabilization time leader election protocol with asymptotically optimal $O(\log \log n)$ number of states.

In Section 6 we highlight two very recent advances: parameterized protocols proposed by Berenbrink *et al.* [16], which allow smooth trade-off between space and time, and fast constant-state protocols proposed by Kosowski and Uznański [37].

2 Overview of population protocols for exact majority and leader election

Since the introduction and the first analysis of population protocols in Angluin *et al.* in [5], a multitude of papers have considered this model and presented solutions to various fundamental problems. The survey by Aspnes and Ruppert [11] discusses variants of population protocols and several early results.

Draief and Vojnović [29] and Mertzios *et al.* [38] focused on population protocols for exact majority and analyzed two similar four-state protocols, which solve the problem on any graph in polynomial time, *w.h.p.* and in expectation. Mertzios *et al.* [38] showed that the expected stabilization time of their protocol is $O(n^5)$ on any graph and $O(n \log n / |a_0 - b_0|)$ on the complete graph. Draief and Vojnović [29] derived the same bound for the complete graphs and obtained also bounds on the expected stabilization time for other graphs, including cycles, stars and random graphs.

Both four-state majority protocols use the same idea of weak versions a and b of the two main opinions A and B. The transition functions of their protocols are given in the table below. The states X and Y stand for distinct strong opinions, x and y stand for the corresponding weak opinions, and q stands for any state.

	(q', q'')	$\delta(q', q'')$
1	(X, Y)	(x, y) in [38], (y, x) in [29]
2	(X, y)	(x, X)
3	(X, x)	(X, x)
4	(x, y)	(x, y) in [38], (y, x) in [29]
5	(q, q)	(q, q)

The strong versions of the two opinions can be seen as tokens moving around in the population. When interacting with a node possessing a weak opinion of opposite sign, the strong opinion converts this weak opinion to a weak opinion of its own kind (transition 2). If two strong opinions A and B interact, they cancel each other, converting to the weak versions a and b (transition 1). The correctness of the protocol follows from the fact that the difference between the number of nodes with (strong) opinion A and B is preserved during the computation and eventually all strong opinions of the initial minority are canceled out. Finally, the remaining strong opinions of the majority convert every weak opposite opinion.

Draief and Vojnović [29] viewed their algorithm as a special case of the interval consensus protocol of Bénézit *et al.* [14], and analyzed it in the continuous time model, which for the case of uniform edge rates is roughly equivalent to the random interaction model.

Angluin *et al.* [7] derived population protocols with constant number of states for various functions. Their protocols are *w.h.p.*-correct and require the presence of the leader from the onset of the computation to synchronize the nodes. Their exact majority protocol *w.h.p.* converges in $O(\log^2 n)$ time and is based on the idea of phases consisting of cancellations and duplications of opinions, which has been used in many subsequent papers. Each phase takes $\Theta(\log n)$ time. The first half of the phase is a cancellation stage, while the second half is a duplication stage. The leader is used to implement a mechanism, which ensures that the nodes progress together from stage to stage. In the cancellation stage of a phase, if two (strong) opposite opinions meet (interact), they cancel each other. In the duplication stage, if a node possessing a strong opinion meets a node without a strong opinion, then both adopt the strong opinion involved in this interaction, i.e., the strong opinion is duplicated. To preserve the majority opinion, the two new tokens are not allowed to duplicate in this phase anymore. Within $O(\log n)$ repetitions of cancellation-duplication phases, all nodes possess the majority opinion. We discuss in more detail this cancellation-duplication framework in Section 4.

Population protocols with constant state space, like the four-state exact-majority protocols, are simple and easy to implement, but they are usually slow. However, the stabilization time of the four-state protocols decreases, if the initial imbalance $|a_0 - b_0|$ is large, so their performance would be improved, if we could boost the initial imbalance. To achieve this, Alistarh *et al.* [4] simply multiplied the initial opinions by some integer $r \geq 2$. Then, the nodes keep the number of strong opinions they currently possess, requiring r states for this. When the strong opinions of the initial minority are canceled out, $|a_0 - b_0|r$ strong opinions of the initial majority are in the population. This speeds up both the canceling of strong opinions and the converting of weak opinions of the initial minority. Refining this idea, Alistarh *et al.* [1] derived an exact majority protocol with stabilization time $O(\log^3 n)$, *w.h.p.* and in expectation, and $O(\log^2 n)$ states.

Bilke *et al.* [20] proved that the cancellation-duplication framework from [7] can be extended to the leaderless case, if the agents have enough states to count their interactions. Their majority protocol has stabilization time $O(\log^2 n)$ *w.h.p.* and in expectation, and uses $O(\log^2 n)$ states. Berenbrink *et al.* [18] extended the previous results on majority protocols to $k \geq 2$ opinions (*plurality voting*) and arbitrary graphs. Their algorithm uses the idea of load balancing [41] and achieves $O(\text{polylog}\, n)$ time using a polynomial number of states, assuming that the initial imbalance bwteen the first and second opinion is $\Omega(n/\text{polylog}\, n)$. For the case of complete graphs and two opinions ($k = 2$), their protocol converges *w.h.p.* in $O(\log n)$ time. Gąsieniec *et al.* [32] derived space optimal population protocols for the absolute and relative majority problem.

Recently Alistarh *et al.* [2] showed that any majority protocol, which has expected stabilization time $O(n^{1-\epsilon})$, where ϵ can be any positive constant, and satisfies the properties of *monotonicity* and *output dominance*. needs $\Omega(\log n)$ states. They also derived an algorithm, with $\Theta(\log n)$ states and stabilization time $O(\log^2 n)$ *w.h.p.* and in expectation. Informally, the running time of a monotonic protocol does not increase if executed with a smaller number of agents. The output dominance means that if the positive counts of states in a stable configuration are changed, then the protocol will stabilize to the same output. All known majority protocols satisfy both these properties, but it is not clear whether they are necessary, so we cannot exclude the possibility that the conditional lower bound $\Omega(\log n)$ on the number of states does not hold in the general case.

Very recently Kosowski and Uznański [37] and Berenbrink *et al.* [16] showed that there are algorithms with polylogarithmic convergence time T_C while using $o(\log n)$ states. As outlined in [15], in [37] the authors design a programming framework with corresponding compilation schemes that provide a simple way of obtaining protocols, which are *w.h.p.* correct, use $O(1)$ states and converge in expected polylogarithmic time. Their protocols can be made always correct by using $O(\log \log n)$ states per node, while keeping polylogarithmic time, or by increasing the time to $O(n^{\epsilon})$, while keeping a constant bound on the number of states. In [16] the authors derive an always correct majority protocol which converges *w.h.p.* in $O(\log^2 n/\log s)$ time and uses $\Theta(s + \log \log n)$ states, as well as an always correct majority protocol which stabilizes *w.h.p.* in $O(\log^2 n/\log s)$ time and uses

$O(s \cdot \log n / \log s)$ states, where $s \in [2, n]$.

Turning attention into the leader election problem, it is easy to come up with a 3-state protocol which elects the leader in complete graphs in $O(n)$ time in expectation. There are two leader states L_1, L_2, and one follower state F, and initially each node starts in state L_1. During the computation each node in a state L_x is a leader. If two leaders of different type interact, the L_2 node becomes a follower. At each interaction, the leader state is switched (L_1 becomes L_2 and vice-versa), unless the leader turns into a follower. Eventually one single leader remains in the population.

Recent $O(\text{polylog}\, n)$-time $O(\text{polylog}\, n)$-state population protocols for leader election have been derived by Alistarh and Gelashvili [3]. Their algorithm uses $O(\log^3 n)$ states and $O(\log^3 n)$ time, *w.h.p.* and in expectation. Doty and Soloveichik [28] showed that any leader election protocol with constant size memory requires at least linear convergence time. Alistarh *et al.* [1] proved that any leader election or exact majority protocol with stabilization time $n/\log^{\omega(1)} n$ requires $\Omega(\log\log n)$ states. In the same paper, they design an algorithm with $O(\log^2 n)$ states and polylogarithmic stabilization time. Bilke *et al.* [20] reduced the expected and *w.h.p.* time to $O(\log^2 n)$ while keeping the number of states at $O(\log^2 n)$. Subsequently, keeping the stabilization time at $O(\log^2 n)$, Alistarh *et al.* [2] reduced the number of states to $O(\log n)$, and finally Gąsieniec and Stachowiak [33] reduced the number of states to asymptotically optimal $O(\log\log n)$.

The papers by Kosowski and Uznański [37] and Berenbrink *et al.* [16] mentioned earlier consider both the exact-majority and the leader election problems. The former shows protocols for leader election which have the same performance characteristics as their exact-majority protocols. The latter develops a parameterized leader election protocol, which gives, for example, a protocol stabilizing *w.h.p.* in $o(\log^2 n)$ time with $O(\log\log n)$ states.

A somewhat different but related line of research is on approximate majority. Angluin *et al.* [8] designed a simple 3-state protocol, which decides majority in $O(\log n)$ time, *w.h.p.*, provided that the initial difference is at least $\omega(\sqrt{n}\log n)$. The protocol is very simple. If two agents of opposite opinion meet, one of them enters a so called "undecided" state. An undecided agent adopts at the next interaction the opinion of its partner. Protocols utilizing the concept of such an undecided state have recently been analyzed in the framework of plurality consensus in a parallel communication model, see e.g. [13, 35, 17, 24].

Another related problem considered in the framework of population protocols is counting. In Mocquerd *et al.* [40], the authors propose a population protocol for computing the difference between nodes being initially in state A and the ones in state B. Their algorithm uses $O(n^{3/2})$ states and requires *w.h.p.* $O(\log n)$ time. For further results on counting agents in various settings and assuming different fairness conditions the reader is referred to [10, 36, 12, 25].

Doty *et al.* [27] consider the size counting problem, i.e., calculating the exact number of agents in a population. They present a uniform algorithm that converges in time $O(\log n \log\log n)$ and requires $O(n^{60})$ states, *w.h.p.* This protocol uses leader election as a sub-protocol, which requires $O(\log n \log\log n)$ time and $O(n^{18})$ states. At the end, they indicate how to reduce the state complexity to $O(n^{30})$ for exact counting and to $O(n^{9})$ for leader election.

Doty and Eftekhari [26] study uniform population protocols for estimating $\log n$ within a constant additive error, or equivalently, estimating the size of the population within a constant multiplicative factor. They present an algorithm, which uses $O(\log^7 n \log\log n)$ states and requires time $O(\log^2 n)$, *w.h.p.* The protocol is converging but not terminating, in the sense that the population does not signal when the population is close to $\log n$. The authors then show that a uniform population protocol for any problem requiring more than constant time cannot terminate with probability bounded away from 0, subject to some assumptions about initial configurations. However, if the leader exists from the beginning of the computation, then the protocol for estimating $\log n$ can be made terminating.

In Sections 4 and 5, we outline recent protocols for exact majority and leader election. Our focus is on the main techniques, which lead to protocols with low space and time requirements.

3 Main probabilistic tools

While analyzing the correctness and time performance of population protocols, we frequently look what happens during a given period of $C \log n$ rounds, for some constant C. The expected number of interactions of a given node during this period is $2C \log n$ and, by a simple application of the Chernoff bounds (see e.g. [39]), *w.h.p.* each node participates in roughly that many interactions. This is formalized in the proposition below, which is used in most analyses, though often more detailed insights into the relative activity of different nodes are needed.

Proposition 1. *For all* $C > 0$ *and* $0 < \delta < 1$, *during a period of* $C \log n$ *rounds, with probability at least* $1 - n^{-\Theta(\delta^2 C)}$, *each node participates in at least* $2C(1 - \delta)\log n$ *and at most* $2C(1 + \delta)\log n$ *interactions.*

Population protocols frequently refer to the (one-way) *epidemics*, or *broadcast* process, which completes *w.h.p.* in $\Theta(\log n)$ rounds. Each node is either in an M state (has the message) or in a $\neg M$ state, and whenever a node in a state M interacts with a node in state $\neg M$, the latter changes to an M state (gets the message). A node can also get the message spontaneously, without receiving it from anyone else. The process starts when the first node gets (spontaneously) the message and completes when all nodes have the message. The proposition below is the basic statement about the probabilistic dynamics of the broadcast process, but, as with Proposition 1, often more detailed properties of this process are needed.

Proposition 2. *There is a constant c_0, such that for $c \geq c_0$, the broadcast process completes in $c \log n$ rounds with probability at least $1 - n^{-\Theta(c)}$.*

4 Exact majority

4.1 Canceling-doubling phases with synchronization

In this section we discuss the framework of canceling-doubling phases, which was introduced by Angluin *et al.* [7] and is the basis of many exact-majority protocols. We assume first that the nodes are synchronized in the following way. For a constant C of our choice, *w.h.p.* all nodes receive the signal "change" every $\Omega(\log n)$ many rounds, with the intervals between the consecutive signals spanning at least $C \log n$ rounds. When a node receives such signal, it changes to the next stage of the computation. The nodes do not receive these signals exactly at the same time, but we assume that *w.h.p.* they all receive the same signal within a period of $c \log n$ rounds, for some constant c which is considerably smaller than C. Assuming, by induction, that all nodes have just entered the current stage of the computation, *w.h.p.* each node will participate in at least $C \log n$ interactions before the nodes start moving on to the next stage (from Proposition 1). While the current stage progresses, the levels of activity of the nodes (the number of their interactions) may start diverging, but the next signal will again tightly synchronize their progress, moving them roughly together to the next stage.

We view the A/B votes as tokens which can have different values (or magnitudes). Initially each node has one token of type A or B with value 1 (the base, or original, value of a token). Throughout the computation, each node either has one token or is *empty*. Let $v.token \in \{A, B, \emptyset\}$ be the type of token held by node v. In the following we say that two tokens *meet* if their corresponding nodes interact.

- When two opposite tokens A and B of the same value meet, then they can cancel each other and the nodes become empty. Such an interaction is called *canceling* of tokens.
- When a token of type $X \in \{A, B\}$ and value z interacts with an empty node, then this token can split into two tokens of type X and half the value $z/2$, and each of the two involved nodes takes one token. We call such an interaction *splitting*, *duplicating* or *doubling* of a token.

We also use the notion of the *age* of a token, which is the number of times it has undergone splitting. Thus the relation between the value z and the age g of a token is $z = 1/2^g$. Note that any sequence of canceling and splitting interactions preserves the difference between the sums of the values of all A and B tokens. This difference is always equal to the initial imbalance. The primary objective is to eliminate all minority tokens.

The computation consists of $O(\log n)$ phases, each phase consists of two stages (the canceling stage followed by the doubling stage), with each stage having $\Theta(\log n)$ rounds and synchronized as described above. When two nodes interact and they are in the same stage, then they attempt the canceling or doubling operation (according to the stage), with the condition that each token can split only once in one doubling stage. Each node v, in addition to $v.token$, stores four boolean flags: $v.stage \in \{canceling, doubling\}$ – the current stage; $v.doubled$ – true, if the node has already doubled in the current phase; $v.done$ – true, if the node has already made the decision on the final output; and $v.fail$ – true, if the node realized that the computation has failed.

From the global point of view, *w.h.p.* each new phase $p \geq 0$ starts with all nodes in normal states ($\neg v.done$ and $\neg v.fail$ for all nodes), all having just entered this phase, and all tokens having the same value $1/2^p$. The phase completes successfully, if at some point all nodes have just moved to the next phase $p + 1$ and are in normal states, and all remaining tokens have the same value $1/2^{p+1}$ (that is, all tokens which survived canceling managed to double in phase p). The difference between the numbers of opposite tokens has doubled and is now equal to $2^{p+1}|a_0 - b_0|$.

The computation *w.h.p.* keeps successfully completing consecutive phases. This is because each canceling stage *w.h.p.* leaves at least $(2/3)n$ nodes empty or eliminates all minority tokens, so if any minority tokens survive a canceling stage, then *w.h.p.* all surviving tokens split in the subsequent doubling stage (*w.h.p.* each token has at least $C \log n$ attempts at splitting and the probability of success in one attempt is at least $1/3$). Each successful phase halves the value of tokens and doubles the difference between A tokens and B tokens, until the *critical phase* p_c, which is the first phase $0 \leq p_c \leq \log n - 1$ when the difference between the numbers of opposite tokens is

$$2^{p_c}|a_0 - b_0| > n/3. \tag{1}$$

The large difference between the numbers of opposite tokens implies that *w.h.p.* all minority tokens will be eliminated in this phase, if they have not been eliminated yet in previous phases. More specifically, at the end of phase p_c, *w.h.p.* only tokens of the majority opinion are left and each of these tokens has value either $1/2^{p_c+1}$ if the token has split in this phase, or $1/2^{p_c}$ otherwise.

If at least one token has value $1/2^{p_c}$, then this token has failed to double during this phase and assumes that the computation has completed. The node with this token enters the *done* state and broadcasts its (majority) opinion to all other nodes. In this case phase p_c is the *final phase*.

Otherwise, if at the end of the critical phase all tokens have value $1/2^{p_c+1}$, then no node knows yet that all minority tokens have been eliminated. The computation proceeds to the next phase $p_c + 1$, which will be the final phase. Phase $p_c + 1$ will start with more than $(2/3)n$ tokens and all of them of the same type, so at least one token will have to fail to double, will enter the *done* state and will broadcast its opinion. In both cases, when either phase p_c or phase $p_c + 1$ is the final one, the failure to double is taken as the indication that *w.h.p.* all tokens of opposite type have been eliminated. Some tokens may still double in the final phase and enter the next phase (receiving later the message that the computation has completed), but *w.h.p.* no node reaches the end of phase $p_c + 2 \leq \log n + 1$. Thus *w.h.p.* within $O(\log^2 n)$ time, all nodes enter the *done* state and know the majority opinion.

The computation fails *w.l.p.*[2] when two nodes enter the *done* state with opposite opinions. If such failure occurs, then all nodes become aware of it and the computation could be restarted (this requires that the nodes store their initial opinions). The computation may also fail, if we cannot exclude the possibility that the synchronization fails and we get tokens in the canceling stages of two different phases. Such tokens would have different values, so if they cancel each other, the difference between the sums of the values of A tokens and B tokens may change, possibly changing its sign (and the minority becomes the majority).

Thus we get a *w.h.p.*-correct protocol, which converges in $O(\log^2 n)$ time *w.h.p.* and in expectation, provided that the required synchronization is implemented. The protocol uses a constant number of states to store the type of token and the flags, plus the states needed to implement synchronization.

In Sections 4.2, 4.3, and 4.4, we outline the following ways of implementing synchronization.

- Synchronization by the leader, which was considered by Angluin *et al.* [7] and gave a *w.h.p.*-correct majority protocol using a constant number of states and converging in $O(\log^2 n)$ time, *w.h.p.*, but requiring a leader node (Section 4.2).
- Synchronization by local counting of interactions, which was considered by Bilke *et al.* [20] and led to an always correct majority protocol using $O(\log^2 n)$ states and stabilizing in $O(\log^2 n)$ time, *w.h.p.* and in expectation (Section 4.3).
- Synchronization by clock nodes, which was considered by Alistarh *et al.* [2] and led to an always correct majority protocol using $O(\log n)$ states and stabilizing in $O(\log^2 n)$ time, *w.h.p.* and in expectation (Section 4.4).

[2]*w.l.p.* – *with low probability* – means that the opposite event happens *w.h.p.*

In Section 4.5, we sketch a protocol proposed recently by Berenbrink *et al.* [15], which overcomes the seemingly inherent $\log^2 n$ asymptotic time complexity of the canceling-doubling framework and stabilizes with the correct majority output within $O(\log^{5/3} n)$ time, *w.h.p.* and in expectation, using $O(\log n)$ states.

4.2 Canceling-doubling phases synchronized by the leader

Angluin *et al.* [7] proposed synchronization of population protocols by a *phase clock* based on an epidemics process controlled by the leader node, assuming that the unique leader exists in the population already in the initial configuration. They considered a broader context of implementing arithmetic operations and relations, including comparison, on values stored in the population in unary. In that context, the input configuration for the majority problem is viewed as a configuraton storing values a_0 and b_0, so the majority problem reduces to the problem of comparing two values.

The phase clock in Angluin *et al.* [7] is implemented in the following way. Each node v stores its clock value $v.h \in H = \{0, 1, 2, \ldots, m-1\}$, where m is a parameter of the clock. This is the cyclical range of the hours of the clock: the next value after $m-1$ is 0 and the difference between two values x and y is equal to $\min\{(x-y) \bmod m, (y-x) \bmod m\}$. The leader starts with value 0 and all other nodes start with $m-1$. Over a sequence of polynomial number of interactions, *w.h.p.* at each step the clock values of the nodes differ by at most $m/3$ and the clock value of the leader is at the front (in the cyclical order of H). That is, while the clock values of individual nodes do not have to stay the same, *w.h.p.* they remain close, occupying a sub-range $\{i, i+1, \ldots, i+j\} \subset M$, for some i and $j \leq m/3$ (the additions are modulo m), with the clock of the leader equal to $i+j$.

This behavior of the phase clock is achieved by the following transitions. When the leader is involved in an interaction and both nodes have the same value, the leader increments its clock. For any other interaction, if the interacting nodes have different but consistent clock values, then the smaller clock value is updated to the larger one. The clock values of two nodes v and u are consistent, if they differ by at most $m/3$, and if one of them is the leader, then its clock value is not smaller (in the cyclical order of H) than the value of the other node.

A node receives the signal to move to the next stage, when its clock value changes to 0. When the leader advances to the next hour, that new hour is communicated to all nodes by the broadcast process. Using the upper bound on the completion time of broadcast given in Proposition 2 and the lower bound that *w.h.p.* $\Omega(\log n)$ time is needed for the message to reach the first n^ϵ nodes, for any constant $\epsilon > 0$, it is proven in Angluin *et al.* [7] that their phase clock satisfies the property stated below. This property implies that the phase clock provides the synchronization of the nodes as was needed in Section 4.1, adding only $O(1)$

states to the canceling-doubling majority protocol, but the presence of the leader is essential.

Proposition 3. *There is a constant c_0 such that for any constants $\alpha > 0$, $c \geq c_0$ $C > c$ and $p > 0$, there exist constants m and C' such that with probability at least $1 - n^{-\alpha}$, the phase clock with the parameter m completes successfully n^p stages: each stage spans between $C \log n$ and $C' \log n$ rounds and all nodes move to the next stage within a period of $c \log n$ rounds.*

4.3 Canceling-doubling phases synchronized by local counting

To avoid the leader, Bilke *et al.* [20] synchronize the nodes in a canceling-doubling majority protocol by making each node keep count of its interactions. More precisely, each node keeps its own count of phases and the count of its steps (interactions) within the current stage of the phase. Thus, in addition to *v.token*, *v.stage*, *v.doubled*, *v.done* and *v.fail*, node v keeps also the following data, requiring in total $\Theta(\log^2 n)$ states.

- *v.phase* $\in \{0, 1, 2, \ldots, \log n + 2\}$ – the counter of phases.
- *v.stage_step* $\in \{0, 1, 2, \ldots, (C \log n) - 1\}$ – the counter of steps in the current stage, where C is a suitably large constant.

Throughout the whole computation, the triple (*v.phase*, *v.stage*, *v.stage_step*) is viewed as the (combined) interaction counter *v.time* $\in \{0, 1, 2, \ldots, 2C \log^2 n)\}$ of node v (assuming that the canceling and doubling stages are numbered 0 and 1). This counter is incremented by 1 at the end of each interaction. If *v.stage_step* becomes 0 after such an increment, then node v “gets the signal” to move to the next stage (that is, to move from canceling to doubling in the same phase, or from doubling to canceling in the next phase).

If nodes were simply incrementing their interaction counters by 1 at each interaction, then the counters would start diverging too much for the canceling-doubling process to work correctly. The following simple mechanism keeps the nodes sufficiently synchronized. When two interacting nodes are in different stages, then the node in the lower stage “gets the signal” to move to the next stage and sets its step counter to the beginning of the next stage. The protocol relies on this synchronization mechanism in the high-probability case when all nodes are in two consecutive stages of the computation (that is, when the counters (*v.phase*, *v.stage*) of any two nodes differ by at most 1). In this case the process of pulling all nodes up to the next phase follows the pattern of broadcast. Once started, the broadcast is completed *w.h.p.* within $c \log n$ rounds, where constant c is sufficiently smaller than C, and all nodes are together in the beginning part of the next stage of the computation. Thus throughout the computation of the protocol, *w.h.p.* all nodes are in two consecutive stages of the computation. If two interacting nodes are not in the same or adjacent stages (a low but positive probability), then their local times (step counters) are considered inconsistent and both nodes enter the special *fail* state.

The computation can fail *w.l.p.*, but now, in contrast to the $O(1)$-state synchronization by the phase clock controlled by the leader, the nodes have sufficient information to realize (eventually) that something has gone wrong. In addition to the failure when two nodes enter the *done* state with opposite-type tokens, the computation also fails when the step counters of two interacting nodes are not consistent, or when one node reaches phase $\log n + 2$. Whenever a node realizes that any of these low-probability events has occurred, it enters the *fail* state and broadcasts this state.

The protocol stabilizes within $O(\log^2 n)$ time *w.h.p.* and in expectation either in the *correct* all-*done* configuration (*w.h.p.*) or in the all-*fail* configuration (*w.l.p.*). This fast protocol, which *w.l.p.* may fail, can be combined with a slow always correct backup protocol to give an exact-majority protocol which requires $\Theta(\log^2 n)$ states per node and stabilizes in the exact majority within $O(\log^2 n)$ time *w.h.p.* and in expectation. The standard technique is to run both the fast and the slow protocols in parallel and make the nodes in the *fail* state adopt the outcome of the slow protocol. The slow protocol runs in expected polynomial time, say $O(n^\alpha)$ time, but its outcome is used only with low probability of $O(n^{-\alpha})$, so it contributes only $O(1)$ to the overall expected time. The four-state majority protocol can be taken as the slow back-up protocol, and we obtain an always correct exact majority protocol which uses $O(\log^2 n)$ states and stabilizes in $O(\log^2 n)$ time *w.h.p.* and in expectation.

4.4 Canceling-doubling phases synchronized by clock nodes

The exact-majority protocol described in Section 4.3 requires $\Theta(\log^2 n)$ states per node. Alistarh *et al.* [2] reduced the number of states to $O(\log n)$, maintaining the $O(\log^2 n)$ bound on the stabilization time. Since their protocol satisfies the conditions of monotonicity and output dominance, in view of the lower bound shown in [2], the $O(\log n)$ number of states is asymptotically optimal for this type of protocols. The logarithmic number of states is achieved by using a *leaderless phase clocks*, which is based on splitting the nodes into *worker nodes* and *clock nodes*. The worker nodes execute the primary task (in our case, they compute the majority in a sequence of canceling-doubling phases), while the clock nodes keep the time. A worker node checks whether it should proceed to the next stage of the computation whenever it interacts with a clock node. A similar idea was used in protocols in other communication models to save memory of the nodes, for example in Ghaffari and Parter [35].

We outline the construction of the leaderless phase clock, following the description of its variant in [15]. A notable difference between the phase clocks in [2] and in [15] is that in [2] the clock nodes keep their time counters synchronized on the basis of the power of two choices in load balancing: when two nodes meet, only the lower counter is incremented. In contrast, in [15] both interacting clock nodes increment their time counters, with the exception that the slower node is pulled up to the next $\Theta(\log n)$-length stage, if the faster node is already there.

The nodes are partitioned into two sets with $\Theta(n)$ nodes in each set. One set consists of the worker nodes, which may carry opinion tokens and work through canceling-doubling phases to establish the majority opinion. These nodes maintain only information on whether they carry any token, and if so, then the value of the token (equivalently, the age of the token, that is, the number of times this token has been split). Each worker node has also a constant number of flags which indicate the current activities of the node (for example, whether it is in the canceling or doubling stage of a phase), but it does not maintain a detailed counter of the steps in the current stage.

The other set consists of clock nodes, which maintain step counters, counting their interactions with other clock nodes modulo $2C \log n$, for a suitably large constant C, and synchronizing with other clocks by the broadcast mechanism at the end of stage (when their counters come back to 0).

The worker nodes interact with each other in a similar way as in the protocol in Section 4.3, but now to progress orderly through the computation they rely on the relatively tight synchronization of clock nodes. A worker node v advances to the next part of the current phase (or to the next phase, or the next epoch), when it interacts with a clock node whose clock indicates that v should progress. There is also a third type of nodes, the *terminator nodes*, which appear later in the computation. A worker or clock node becomes a terminator node when it enters a *done* or *fail* state. The meaning and function of these special states are as defined in previous subsections.

A standard input instance, when each node is a worker with a token of value 1, is converted into a required initial workers-clocks configuration during the following $O(\log n)$-time preprocessing. When two value-1 tokens of opposite type interact they cancel out and one of the two involved nodes, say the one which has had the token B, becomes a clock node. If two value-1 tokens of the same type interact and their step counters have different parity, then the tokens are combined into one token of value 2. The combined token is taken by one node, while the other node, say the one with the odd counter, becomes a clock node. All nodes count their interactions during the preprocessing, but the $O(\log n)$ states needed for this are re-used when the preprocessing completes. At this point the worker nodes have an input instance with the base value of tokens equal to 2. Some tokens may have value 1 (and can be considered as if already split in the first phase) and some nodes may be empty.

The analysis of the execution of the canceling-doubling protocol with leaderless phase clock follows closely the analysis of the protocol with all nodes counting their interactions.

4.5 Majority protocol with $O(\log^{5/3} n)$ time and $O(\log n)$ states

Berenbrink *et al.* [15] present a majority population protocol with stabilization time $O(\log^{5/3} n)$ *w.h.p.* and in expectation and $O(\log n)$ states. Note that no majority protocol with $O(\mathrm{polylog}\, n)$ states and running time $O(\log^{2-\alpha} n)$, for any constant $\alpha > 0$, has been known before, not even if the weaker notions of the "convergence time" or "*w.h.p.* correctness" were considered.

Most known majority protocols with polylogarithmic number of states and convergence time are based on the idea of a sequence of $O(\log n)$ phases of canceling and doubling stages, described in Section 4.1. Each stage has length $\Omega(\log n)$ and the nodes are synchronized when they proceed from stage to stage. The protocol in [15] also uses this overall canceling-doubling framework but with *shorter phases* of length $\Theta(\log^{2/3} n)$ each, at the expense of weakening synchronization. The previous majority protocols cease to function properly with sub-logarithmic phases. Such phases are just too short to synchronize nodes, resulting in tokens from different phases existing in the system at the same time. The computation can then potentially get stuck, since opposite-opinion tokens from different phases cannot cancel each other. Moreover, we do not even have the guarantee that every node will be activated at all during a short phase – in fact, we know that *w.h.p.* some nodes will not. The previous protocols require that *w.h.p.* each node is activated at least logarithmically many times during each phase.

Berenbrink *et al.* [15] devise a way to deal with the nodes which advance too slowly or too quickly through the short phases, that is, the nodes whose progress is out of sync with the main bulk of nodes. Their protocol groups $\log^{1/3} n$ consecutive phases in one $\Theta(\log n)$-time *epoch*. The configuration of the system remains reasonably tidy throughout one epoch even without explicit synchronization. At the end of the epoch, a $\Theta(\log n)$-time "cleaning-up" period is added to let the nodes synchronize their progress before they start moving on to the next epoch. There are still $O(\log n)$ phases, so there are $O(\log^{2/3} n)$ epochs giving the $O(\log^{5/3} n)$ bound on the convergence time of the protocol. Regarding the number of states, it is first shown in [15] that if each node keeps track of its interactions, to support synchronization of the type described in Section 4.3, then $\Theta(\log^2 n)$ states per node are needed: the product of $\Theta(\log^{2/3} n)$ states to keep track of epochs, $\Theta(\log n)$ states to keep track of the steps in the current epoch and $\Theta(\log^{1/3} n)$ to store the relative age of a token (relative to the beginning of the current epoch).

The nodes store the relative ages of tokens because the possibility of out-of-sync nodes means that the number of interactions a node has had so far in the current epoch does not indicate the age of a token in this node. [15] explains how the number of states can be reduced to $\Theta(\log n)$ using the leaderless phase clock outlined in Section 4.4.

During the execution of the protocol, we have a small but *w.h.p.* positive number of out-of-sync tokens, which move to the next phase either too early or too late (with respect to the expectation) or simply do not succeed with splitting within a short doubling stage. Such tokens stop contributing to the regular dynamics of canceling and doubling. The $\Theta(\log n)$-time cleaning-up period at the end of each epoch enables the out-of-sync tokens to reach the age required at the end of the epoch, and then gives time for synchronizing all nodes, by the broadcast process, when they are moving to the next epoch. The analysis of the progress of tokens through the phases of the same epoch considers separately the tokens which remain synchronized and the out-of-sync tokens.

It is shown that *w.h.p.* there are only $O(n/2^{3\log^{1/3} n})$ out-of-sync tokens in any one epoch. *W.h.p.* all out-of-sync tokens in the current epoch reach the correct age of $\log^{1/3} n$ (correct for the end of this epoch w.r.t. the beginning of the epoch) by the midpoint of the cleaning-up period, for each epoch until the *final epoch* j_f. The main reason why in each epoch before the final one all out-of-sync tokens *w.h.p.* manage to catch up with the expected progress is that these tokens can create (by splitting) at most $O(n/2^{3\log^{1/3} n}) \times 2^{\log^{1/3} n} = o(n)$ tokens. All these tokens can easily be accommodated in the nodes because the computation maintains the invariant that there are $\Omega(n)$ empty nodes.

In the final epoch at least one out-of-sync token completes the epoch without reaching the required age. When the system completes the final epoch, the task of determining the majority opinion is not fully achieved yet. In contrast to the generic protocol described in Section 4.1, where on completion of the final phase *w.h.p.* only majority tokens are left, in the $O(\log^{5/3} n)$-time protocol, there may still be a small number of minority tokens at the end of the final epoch, so some further work is needed.

A node which has failed to reach the required age by the end of the current epoch, discovering that way the final epoch, enters the *additional_epoch* state and broadcasts this state through the system to trigger an *additional epoch* of $\Theta(\log^{1/3} n)$ phases. More precisely, the additional epoch consists of at most $3\log^{1/3} n$ phases corresponding to epochs $j_f - 1$ (if $j_f > 0$), j_f and $j_f + 1$. Each phase in the additional epoch has a full length of $\Theta(\log n)$ rounds. *W.h.p.* these phases include the critical phase p_c and the phase $p_c + 1$, defined by (1). The computation during the additional epoch is as described in Sections 4.1 and 4.3, taking $O(\log^{5/3} n)$ time to reach the correct all-*done* configuration (*w.h.p.*) or the all-*fail* configuration (*w.l.p.*).

Implementing this protocol with the leaderless phase clock described in Section 4.4 and combining with the back-up four-state protocol give an always correct exact majority protocol which uses $O(\log n)$ states and stabilizes in $O(\log^{5/3} n)$ time *w.h.p.* and in expectation.

5 Leader election

5.1 A general framework

Most recent population protocols for the problems we are interested in implement first some mechanism to allow the nodes the access to an (almost) fair random coin. In many cases, this is achieved in the following way. At the beginning, each node initializes a so called coin bit with 0, and this bit is flipped at every interaction. It can be shown (see e.g. [1]) that by the time all nodes performed at least four interactions, the nodes are divided almost evenly into 0- and 1-nodes, and this property is maintained for polynomial number of steps, *w.h.p.*. This implies that at every further interaction, each node meets a 0- or 1-node with probability $(1 \pm o(1))/2$.

Several protocols for leader election usually consist of two main parts. In the first part, each node computes a certain value, which often follows a geometric distribution. In the second part, these values are broadcasted within the population by some simple epidemic process, see Proposition 2. If a node observes a value that is larger than its own, then this node turns into a so called follower or minion (i.e., it can not become a leader anymore), and helps broadcasting the largest value it has seen so far. Thus, only nodes that computed the maximum value in the first phase can become leaders. Unfortunately, there may be many nodes with this maximum value. In order to decide which of these nodes should become a leader, different techniques have been applied.

While there have been designed population protocols for leader election with polylogarithmic number of states before (see Section 2), we start our presentation in Subsection 5.2 with two algorithms, which follow the framework described above and require $O(\log^2 n)$ states only. In Subsection 5.3 we describe two follow-up protocols, which work with $O(\log n)$ states. Finally, in the last subsection we present a breakthrough in this area, and give an algorithm with optimal number of states $O(\log \log n)$.

5.2 Protocols with $\Theta(\log^2 n)$ states

In [1] the authors first derive a lower bound $\Omega(\log \log n)$ on the number of states for leader election and exact majority. Then, for both problems a population protocol is presented with $O(\log^2 n)$ states and polylogarithmic time.

Some of the ideas of this leader election protocol have already been used in [3]. As mentioned above, the algorithm itself consists of two main parts: the lottery stage and the competition stage. At the beginning, each node uses its first four interactions to divide the population into two sets, so called 0- and 1-nodes as described above. After all nodes have performed four interactions the two sets have almost equal size *w.h.p.*, see previous subsection. Note that this 0 or 1 value of a node is switched at every further interaction of this node. After its fourth interaction, each node switches to the lottery mode. In this mode, a node computes its so called payoff. That is, it counts the number of interactions with a 1-node until it meets a 0-node, or the number of its interactions reaches a predefined value $\Theta(\log n)$. This is then the payoff of the node. Once the payoff is determined, the node switches to the competition mode to elect the leader.

In the competition mode, each node is at the beginning a contender and initializes its so called level with value 0. The competition mode of a node consists of consecutive phases, each of length $\Theta(\log \text{payoff})$. If in a phase, a node only meets agents with coin value 1, then it increments its level until some predefined value $\Theta(\log n)$ is reached. At each interaction of two nodes, the one with higher state (payoff, level, coin) wins. That is, the node with lower payoff becomes a follower (if it was a contender); if the payoff values are the same, then the node with the lower level turns to a follower. If both values – payoff and level – are the same, then among two contenders the contender with coin value 0 is turned into a follower, if the other contender has coin value 1. Note that if one of the interacting nodes is a follower, then the coin value does not decide the tournament of the two. Each follower always sets its own tuple (payoff, level) to the largest one seen so far. The authors of [1] show the following result.

Theorem 1. *The algorithm described above always elects a unique leader by using $O(\log^2 n)$ states and polylogarithmic stabilization time*[3]*, w.h.p. and in expectation.*

Note that the algorithm above is not uniform.

While keeping the number of states at $O(\log^2 n)$, the parallel time has been reduced to $O(\log^2 n)$ in [20]. In the following, we describe the major ideas of the algorithm, and state the main result. Some building blocks of this algorithm are known from previous work.

The algorithm is divided into two main parts – a leader election part and a verification part. These two parts consist of several phases. The first part itself is divided into two halfs: preprocessing and tournament. In the preprocessing, the nodes are divided into four subsets, $(A, 0)$, $(A, 1)$, $(B, 0)$ and $(B, 1)$. It is guaranteed

[3]The expected time is $O(\log^{5.3} n \log\log n)$ and the *w.h.p.* time is $O(\log^{6.3} n \log\log n)$.

that all these sets have linear size in n, with high probability. In a first phase, the nodes are divided into sets A and B, and then both sets are separated into 0- and 1-nodes. Between these phases, as well as between the preprocessing part and the tournament part, we deploy so called synchronization phases of length $\Theta(\log n)$. In these synchronization phases, the nodes simply wait (i.e., they do nothing, except counting the number of interactions they performed so far). The length of the set generation phases is $\Theta(\log n)$ as well. This ensures that *w.h.p.* at each time step of the preprocessing all nodes are:

- either in the phase of creating A and B nodes, or in the subsequent synchronization phase
- either in the synchronization phase between the set creation phases, or in the phase of creating $0/1$ nodes
- either in the phase of creating $0/1$ nodes or in the subsequent synchronization phase.

Once the number of interactions of a node in a phase reaches the value set for the length of a phase, the node switches to the next phase, and performs the state transitions defined in the protocol for that phase (see [20] for the details).

The synchronization phase between the preprocessing and the tournament ensures that *w.h.p.* at the time the first node enters the tournament part, all the others are divided into the sets defined above. The sets $(A, 0)$, $(A, 1)$, $(B, 0)$, and $(B, 1)$ ensure that the nodes are able to perform Bernoulli trials. In the first phase of this tournament part – we call trial phase – the nodes only count interactions with A-nodes and an interaction with an $(A, 1)$-node is counted as successful. Once the number of (not necessarily successful) counts of a node reaches a predefined value $\Theta(\log n)$, this node leaves this phase and stores the count of its successes. Additionally, for independency reasons in the analysis, the count of the B-nodes is elevated by some large logarithmic value. A subsequent synchronization phase ensures that *w.h.p.* all nodes leave the trial phase before the actual tournament begins. In the tournament, nodes start broadcasting their values of success, and the maximum value wins. That is, at each interaction every node transmits the largest value it has seen so far to its interaction partner, and if a node observes some value that is larger than its own, then it turns to a follower. This phase has again length $\Theta(\log n)$ to ensure that the maximum value in the system is broadcasted to all nodes *w.h.p.* (see Proposition 2). If a node at the end of this phase has not seen a value, which is larger than its own, then this node declares itself a leader candidate, and switches to the verification part.

In the verification part, each node passes through $\Theta(\log n)$ phases, each of length $\Theta(\log n)$. When a leader candidate switches from one phase to the next, it sends out a message, denoted by 0 or 1, depending on whether its interaction partner at this time step is an A- or B-node. Then, the leader candidate waits $C \log n$ interactions, where C is a large constant, before it triggers the next phase. The followers act as relay nodes, by passing messages to their communication partners at each interaction at the beginning of the phase. If two followers possessing different types of messages interact, then they assume that more than one leader candidate is in the system. In such a case, these nodes restart the whole procedure by broadcasting a restart message.

As mentioned above, the followers participate in spreading messages at the beginning of the phase only. After $c \log n$ interactions, where c is a proper small constant (but larger than the constant in Proposition 2), a follower does not pass any message further, and after $3C/4 \cdot \log n$ iterations it forgets the message. These values ensure that *w.h.p.* no follower possesses any message when the next phase is triggered by some leader candidate. It is shown in [20] that if two (or more) leader candidates are in the system, then with constant probability this situation will be detected in an (arbitrary but fixed) phase. Since there are $\Theta(\log n)$ phases, *w.h.p.* the procedure will be restarted if more than one node declared itself a leader candidate. If a leader candidate passes through all the $\Theta(\log n)$ phases, without experiencing a restart, then the leader candidate declares itself a leader, and sends out a message to the system that a leader has been found.

Clearly, with a small probability it may happen that two or more candidates declare themselves leaders. However, in such a case two leaders eventually meet. Then, one of the leaders turns into a follower (by applying standard tie-breaking techniques), and *w.h.p.* in polynomial number of steps, only one leader remains in the system. In [20] the following theorem has been shown.

Theorem 2. *There is an always correct leader election protocol that requires $O(\log^2 n)$ states and has stabilization time $O(\log^2 n)$, w.h.p. and in expectation.*

This protocol is also not uniform, as the nodes must be aware of some $\Theta(\log n)$ values in advance.

5.3 Protocols with $\Theta(\log n)$ states

In [2], Alistarh *et al.* mainly concentrate on exact majority. However, they also present a leader election protocol with $O(\log n)$ states and $O(\log^2 n)$ time. In this paper, they deploy a distributed phase clock and use the coin flip generation technique as in [1]. The nodes are divided into clock nodes and workers as in the exact majority protocol described in the same paper, see Subsection 4.4. Their clock creation mechanism ensures that the number of clock nodes never exceeds $n/2$. At each clock node, the clock is circular and has length $\Theta(\log n)$. Furthermore, it is guaranteed that all clock nodes are within some range ρ around the mean clock value, *w.h.p.*, where ρ is still $\Theta(\log n)$, but much smaller than the length of the clock. The state transitions of the clock nodes to update their clock values is exactly the same as in the corresponding exact majority algorithm, see previous section.

The worker nodes can be contenders or followers (initially all of them are contenders). Each such node is in a certain phase and possesses a so called High/Low value set to 1 or 0 (initialized with 0). The total number of phases is $\Theta(\log n)$. The clock of a clock node is divided into four equal sized segments, and the first and third segments are called ODD and EVEN, respectively. If a contender in an odd (even) phase meets a clock node in an EVEN (ODD) segment, then the contender switches to an intermediate state. At the next interaction, it increments its phase by 1 (unless it is already in the highest phase, predefined at some value $\Theta(\log n)$), and sets its High/Low indicator to the coin value of the interaction partner. If two workers meet and one of them is a contender with a smaller (phase, High/Low value) pair, then the contender becomes a follower and adopts the phase and High/Low value of the other node. The followers always keep the highest (phase, High/Low value) they observe. This way, at each time the highest (phase, High/Low value) pair is spread among the workers and all nodes with smaller (phase, High/Low value) pair are turned into followers (cf. Proposition 2). The authors of [2] show the following theorem.

Theorem 3. *The algorithm above elects a unique leader by using $O(\log n)$ states and has stabilization time $O(\log^2 n)$, w.h.p. and in expectation.*

As the nodes have to know some value $\Theta(\log n)$ in advance, this algorithm is not uniform as well.

In [19], Berenbrink *et al.* present a simple protocol, which has the same state and time guarantees as the one in Theorem 3. This algorithm basically combines the synthetic coin technique of [1] with the tournament phase of [3]. First, each node performs $\Theta(\log\log n)$ interactions, just flipping its coin bit at each interaction. Then, the so called marking phase starts: for another $\log\log n$ interactions, if a node only meets nodes with coin bit value 1, then this node becomes marked. This implies that *w.h.p.* there will be $\Theta(n/\log n)$ marked nodes.

In the tournament phase, every node starts as a leader contender and as long as it is not turned into a follower, it counts its interactions with marked nodes (until a predefined value $\Theta(\log n)$ is reached). This number is called the counter of the node. The counters are broadcasted in the population (at each interaction the nodes exchange their counter values), and if a leader contender with some counter value meets a node with a higher counter, then the leader contender turns to a follower. Each follower keeps the highest counter value ever seen, and spreads this value further at each interaction. If two leader contenders with the same counter but different coin bit value meet, then the leader contender with coin bit value 0 turns into a follower.

5.4 A protocol with optimal number of states

A breakthrough in this field was achieved by Gąsieniec and Stachowiak in [33]. Their algorithm builds partially on the methods of Angluin *et al.* [5] who make use of a phase clock. The phase clock of Angluin *et al.* requires, however, a distinguished node in the system at the beginning, see Subsection 4.2. They mention that the role of this distinguished node can also be fulfilled by a junta of size n^{ϵ}. Probably the most interesting part of the paper of Gąsieniec and Stachowiak is the construction of a junta of size $O(\sqrt{n \log n})$ and the integration of the junta election process into the leader election procedure. These two building blocks are combined in a fascinating way.

The algorithm utilizes directed state transitions. We only present here the main ideas of the algorithm, for the details we refer the reader to [33]. The phase clock itself is very simple. The junta nodes are called leaders and the others are called followers. First, the authors define a circular order on a clock with m values, where we may assume w.l.o.g. m is odd. Assign the numbers $0, \dots, m-1$ to an analogue clock, which can be seen as hours on the dial of this clock. Then $x \leq_m y$ iff the number of hours between x and y is less than $m/2$ in the clockwise order from x to y (e.g. $m-2 <_m 1$ if $m \geq 7$).

If a follower with value x is called by a node with value y, then the follower adopts the value $\max_m\{x, y\}$, where $\max_m$ is the maximum according to the circular order defined above. If a leader with value x is called by a node with value y, then the leader adopts the value $\max_m\{x, (y+1) \bmod m\}$.

In [33], the authors utilize two nested clocks. The inner clock of a node operates at each interaction of this node as described above. The outer clock ticks once directly after the inner clock of the node passes through 0. The authors show that if all agents of the population start the phase clock protocol from 0 when a junta of size $n^{1-\epsilon}$ is elected, then for a polynomial number of rounds, all nodes are between $3m/4$ and $m/4$ when a node passes through 0, and the number of interactions between two subsequent passes through 0 for any agent is $O(n \log n)$, *w.h.p.*

The junta election process is independent of the clocks; however, the junta is then used to implement the phase clock. In order to elect a junta, the authors use a very simple procedure. The states are represented by (l, a), where l is a so called level and a is 0 or 1, where nodes with $a = 0$ do not change their state anymore w.r.t. the junta election process. If some node v in state $(l, 1)$ interacts as a responder with an agent, then its level increases by 1 if the level of the initiator is at least l (a remains 1), and a is set to 0 (l remains the same) if the level of the initiator is less than l. The authors show that *w.h.p.* the highest level a node reaches is $O(\log\log n)$ and there are $O(\sqrt{n\log n})$ nodes at highest level. A subset of these highest level nodes will form the junta, which is used to implement the correct phase clocks. Clearly, the nodes at highest level do not know that they are in the junta; however, this is also not required for the correctness of the overall leader election protocol.

The leader election scheme is as follows. At the beginning, all nodes are leader candidates. Each node performs first junta election, and if its a value is set to 0, it starts with leader election. Nodes with $a = 0$ participate in running the phase clock, where the leader candidates form the junta. Clearly, different nodes may have different level values when their a value is set to 0, so they may run phase clocks at different rates. If a node running the phase clock at some level interacts with an agent running a phase clock on a higher level, then this node becomes a follower (if it was a leader candidate), and it adjusts its level to the level of the interaction partner. The internal and external clocks are reset to 0.

While the phase clocks are running, after the inner clock of a node passes through 0 and this node interacts with a non-leader, the node flips a coin to obtain a 0 or 1 (the coin flip can be implemented by the random interactions, see above). Message 1 is broadcasted together with the level of the corresponding node (cf. Proposition 2), and if a leader candidate with value $(l, 0)$ receives $(l', 1)$ with $l' \geq l$, then this candidate turns into a follower spreading $(l', 1)$ further. This is repeated $\Theta(\log n)$ times (i.e., until value m of the outer clock is reached). The authors show that *w.h.p.* this procedure elects a unique leader. The state and time requirements are $O(\log\log n)$ and $O(\log^2 n)$, *w.h.p.*, respectively.

We should note that the protocol described above is uniform; however, it may fail to elect a leader with some small probability. In order to ensure that the protocol always elects a unique leader, the uniformity condition has been droped. For the details, see [33]. In a subsequent work, Gąsieniec *et al.* [34], extend their algorithm to improve the expected running time to $O(\log n \log\log n)$; however, the *w.h.p.* stabilization time remains $O(\log^2 n)$. Taking into account that there is a matching lower bound of $\Omega(\log\log n)$ w.r.t. the number of states [1], the algorithm of Gasieniec and Stachowiak is state optimal.

6 Further results and open problems

Two very recent papers by Berenbrink *et al.* [16] and Kosowski and Uznański [37], deal specifically with population protocols for exact majority and leader election, focussing on small sate space and fast convergence time. We outline their results.

Berenbrink *et al.* [16] present parameterized protocols, which allow smooth trade-off between space and time, and discuss various extensions of those protocols in order to obtain uniformity. While focusing on fast convergence, for certain cases they also analyze the stabilization time.

The trade-off protocols in [16] are parametrized by a value $s \geq 1$. First they derive a simple protocol that *w.h.p.* correctly solves exact majority in time $O(\log^2 n / \log s)$ and uses $O(s + \log\log n)$ states. To obtain always correct protocols, the well known technique of combining the fast *w.h.p.*-correct protocol with a slow always correct protocol is applied (see Section 4). The obtained hybrid protocol is guaranteed to always converge to the correct output, while preserving the convergence time and the number of states as they are in the *w.h.p.*-correct protocol. To obtain the same time guarantee for stabilization, the number of states was extended to $O(s \log n / \log s)$.

The *w.h.p.*-correct protocol for exact majority proposed in [16] uses a similar phase clock as given in [33], and the nodes essentially perform load balancing. Assuming that the nodes have access to a clock (which is guaranteed *w.h.p.* by the phase clock described in the previous section), they perform three different types of actions: load expansion, load balancing, and synchronization. The protocol operates in phases of length $\Theta(\log n)$. Load expansion means that at its first interaction as an initiator, a node u multiplies its load (set initially to 1 or -1, depending on the initial opinion of the node) by s. During load balancing ($\Theta(\log n)$ interactions), if two nodes interact, then they simply balance their load as evenly as possible. During synchronization, the clocks become synchronized.

One of the main ingredients of the protocol is the implementation of the phase clock. They basically use the phase clock from Gąsieniec and Stachowiak [33], but modify the junta internals and the interplay between the junta and the clocks. Firstly, once an agent leaves the junta election process, it no longer stores the level which it has reached. That way the agents can reuse the states from the junta election in the subsequent load balancing computations. This reusing of states means that the protocol works with only $O(s + \log\log n)$ states rather than $O(s \log\log n)$. Secondly, some further modifications of the Gąsieniec and Stachowiak algorithm are needed to decrease the running time below $\Theta(\log^2 n)$. Instead of comparing just one random bit in a phase of length $\Theta(\log n)$, the nodes sample $\log s - \log\log s$ bits. Then, a one way epidemic is used to broadcast the maximum to all nodes, and only the nodes with this maximum value remain leader contenders after the end of the phase.

In [16] the authors also discuss the uniformity of the population protocols designed so far for exact majority and leader election. While the leader election protocol of [33] can be modified to become uniform, [16] provides the first uniform exact majority algorithm, which stabilizes in sublinear time. This protocol uses $O(s \log n \log\log n / \log s)$ states and stabilizes in $O(\log^2 n / \log s)$ time, *w.h.p.*

Kosowski and Uznański [37] implement a different type of nested phase clocks which use only constant number of states. The resulting algorithms for exact majority and leader election *w.h.p.* converge to the correct output. Further extensions of the algorithms result in always correct protocols at the cost of increasing the number of states to $O(\log\log n)$. The essential building block of their protocols is the implementation of the nested phase clocks. The progress of the clocks is guaranteed by the asynchronous random scheduler. There is a distinguished state X, and the phase of the clock progresses, when the number of agents in state X is in the range $[1, n^{1-\epsilon}]$, where ϵ is a (fixed) parameter of the protocol. The hierarchy of the clocks is implemented in a similar way as the 2-nested clock of [33]. In the clock hierarchy $C^{(1)}, C^{(2)}, \ldots,$ clock $C^{(j)}$ performs at least $r^{(j)} - O(1)$ cycles during one cycle of clock $C^{(j+1)}$, where $r^{(j)}$ is the rate of clock $C^{(j)}$. This way, a clock at hierarchy level $j+1$ has rate $r^{(j+1)} = \Theta(r^{(1)}r^{(j)})$.

For the base clock $C^{(1)}$, it is important to have the correct number of nodes in state X (see above), and to ensure that some nodes remain in this state for a long enough period. Gąsieniec and Stachowiak [33] used a junta for this, and reduced the number of nodes to the correct size by letting the nodes traverse $O(\log\log n)$ levels, requiring $O(\log\log n)$ states. Kosowski and Uznański [37] use the idea of a self-stabilizing oscillator from [30]. At the beginning, all agents are in state X and their number is reduced within polylogarithmic number of rounds to the correct size. Then, the clock works correctly for another polylogarithmic time, which is enough *w.h.p.* to elect a leader or to compute exact majority. During this time, the rate of the base clock is $\Theta(\log n)$ and the rate of the clock at level i is $\Theta(\log^i n)$.

The *w.h.p.*-correct majority protocol in [37] keeps iterating the execution of $\Theta(\log n)$ canceling-doubling phases. Each iteration starts with the initial opinions of the nodes and takes $O(\text{polylog}\, n)$ time. Initially the iterations might not give correct outputs, but the first iteration after the hierarchy of clocks gets initialized gives *w.h.p.* correct majority output. The clocks will stop operating at some point, but the correct output is reached (*w.h.p.*), if they work for long enough to synchronize one full iteration.

Note that in this implementation the number of nodes in X might be reduced to 0, which in turn leads to the possibility that protocols are not always correct. To obtain an always correct majority protocol, phase clocks which do not stop working should be used, for example the $O(\log\log n)$-state clock from Gąsieniec and Stachowiak [33]. An alternative perpetual clock proposed in [37] uses only constant number of states and reaches the required rate within $O(n^{\epsilon})$ rounds for an arbitrary constant $\epsilon > 0$ (smaller ϵ means larger number of states). Moreover, an always correct majority protocol cannot start each new iteration with the initial opinions, as otherwise we will keep getting occasionally incorrect iterations. To prevent this, each iteration starts with the opinions the nodes had after the initial canceling stage of the previous iteration. Those initial canceling stages gradually eliminate the minority opinion, and when it is completely eliminated, all subsequent iteration must return the correct output.

The following results are shown in [37].

- There are population protocols for exact majority and leader election, which are *w.h.p.* correct, and require polylogarithmic time and $O(1)$ states.
- There are always correct population protocols, which solve exact majority and leader election *w.h.p.* in time $O(n^\epsilon)$, for any constant $\epsilon > 0$, and use $O(1)$ states.
- There are always correct population protocols, which solve exact majority and leader election, have *w.h.p.* polylogarithmic convergence time, and use $O(\log\log n)$ states.

There are open questions left regarding the running time of population protocols for for exact majority and leader election. Almost all protocols with polylogarithmic number of states designed so far for these two problems have *w.h.p.* convergence time $\Omega(\log^2 n/\log\log n)$. Gąsieniec *et al.* [34] reduce the expected time of leader election to $O(\log n \log\log n)$, but the bound on the *w.h.p.* time remains $O(\log^2 n)$, and Berenbrink *et al.* [15] decrease the expected and *w.h.p.* running time of exact majority to $O(\log^{5/3} n)$. A natural open question is how close can we get the convergence or stabilization time (*w.h.p.* and/or expected) towards the $O(\log n)$ bound, while keeping the number of states small, ideally optimal.

An interesting open question is whether the $\Omega(\log n)$ lower bound on the number of states of fast exact-majority protocols can be shown if the monotonicity or output dominance assumptions are dropped. Finally, it is still open whether the $\Omega(\log\log n)$ bound on the number of states holds also if one is interested in polylogarithmic convergence time instead of stabilization time.

References

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *Proceedings of the* 28*th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'17*, pages 2560–2579, 2017.

[2] Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'18*, pages 2221–2239, 2018.

[3] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proceedings of the* 42*nd International Colloquium on Automata, Languages, and Programming, ICALP'15, Part II*, pages 479–491, 2015.

[4] Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings*

of the 2015 ACM Symposium on Principles of Distributed Computing, PODC'15, pages 47–56. ACM, 2015.

[5] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[6] Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing, PODC'06*, pages 292–299, 2006.

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

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

[9] Dana Angluin, James Aspnes, David Eisenstat, and Erik Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[10] James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. Time and Space Optimal Counting in Population Protocols. In *Proceedings of the* 20*th International Conference on Principles of Distributed Systems, OPODIS'16*, volume 70, pages 13:1–13:17, 2017.

[11] James Aspnes and Eric Ruppert. An introduction to population protocols. In Benoît Garbinato, Hugo Miranda, and Luís Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer-Verlag, 2009.

[12] Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In Yoram Moses, editor, *Proceedings of the* 29*th International Symposium on Distributed Computing, DISC'15*, pages 631–646, 2015.

[13] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. Plurality consensus in the gossip model. In *Proceedings of the* 26*th Annual (ACM-SIAM) Symposium on Discrete Algorithms, SODA'15*, pages 371–390, 2015.

[14] Florence Bénézit, Patrick Thiran, and Martin Vetterli. Interval consensus: From quantized gossip to voting. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'09*, pages 3661–3664. IEEE, 2009.

[15] Petra Berenbrink, Robert Elässser, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with $O(\log^{5/3} n)$ stabilization time and $O(\log n)$ states. In *Proceedings of the 32nd International Symposium on Distributed Computing, DISC'18*, 2018.

[16] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Domink Kaaser Peter Kling, and Tomasz Radzik. Majority & stabilzation in population protocols. *CoRR*, abs/1805.04586, 2018.

[17] Petra Berenbrink, Tom Friedetzky, George Giakkoupis, and Peter Kling. Efficient plurality consensus, or: The benefits of cleaning up from time to time. In *Proceedings of the* 43*th International Colloquium on Automata, Languages and Programming, ICALP'16*, pages 136:1–136:14, 2016.

[18] Petra Berenbrink, Tom Friedetzky, Peter Kling, Frederik Mallmann-Trenn, and Chris Wastell. Plurality consensus via shuffling: Lessons learned from load balancing. *CoRR*, abs/1602.01342, 2016.

[19] Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and efficient leader election. In *Proceedings of the First Symposium on Simplicity in Algorithms, SOSA'18*, pages 9:1–9:11, 2018.

[20] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC'17*, pages 451–453, 2017. Full version available at arXiv:1705.01146.

[21] Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: on the minimal size of population protocols. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS'18*, pages 16:1–16:14, 2018.

[22] Luca Cardelli and Attila Csiksz-Nagy. The cell cycle switch computes approximate majority. *Nature Scientific Reports*, 2:656, 2012.

[23] Yuen-Jyue Chen, Neil Dalchau, Niranjan Srnivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controlers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.

[24] Andrea Clementi, Mohesen Ghaffari, Luciano Gualà, Emanuele Natale, Francesco Pasquale, and Giacomo Scornavacca. A tight analysis of the parallel undecidedstate dynamics with two colors. In *Proceedings of the* 43*rd International Symposium on Mathematical Foundations of Computer Science, MFCS'18*, pages 28:1–28:15, 2018.

[25] Gennaro Cordasco, Luisa Gargano, Paul Spirakis, and Phillipas Tsigas. Space-optimal proportion consensus with population protocols. In *Proceedings of the* 19*th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS'17*, pages 384–398, 2017.

[26] David Doty and Mahsa Eftekhari. Efficient size estimation and impossibility of termination in uniform dense population protocols. *CoRR*, abs/1808.08913, 2018.

[27] David Doty, Mahsa Eftekhari, Othon Michail, Paul Spirakis, and Michail Theofilatos. Exact size counting in uniform population protocols in nearly logarithmic time. *CoRR*, abs/1805.04832, 2018.

[28] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *Proceedings of the* 29*th International Symposium on Distributed Computing, DISC'15*, 2015.

[29] Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. In *Proceedings of the* 29*th IEEE International Conference on Computer Communications, INFOCOM'10*, pages 1792–1800. IEEE, 2010.

[30] Bartlomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *Proceedings of the* 50*th Annual ACM Symposium on Theory of Computing, STOC'18*, pages 87–99, 2018.

[31] Javier Esparza, Pierre Ganty, Jérome Leorux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017.

[32] Leszek Gasieniec, David Hamilton, Russel Martin, Paul Spirakis, and Grzegorz Stachowiak. Deterministic population protocols for exact majority and plurality. In *Proceedings of the* 20*th International Conference on Principles of Distributed Systems, OPODIS'16*, pages 14:1–14:14, 2017.

[33] Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *Proceedings of the* 29*th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'18*, pages 2653–2667, 2018.

[34] Leszek Gasieniec, Grzegorz Stachowiak, and Przemyslaw Uznanski. Almost logarithmic-time space optimal leader election in population protocols. *CoRR*, abs/1802.06867, 2018.

[35] Mohsen Ghaffari and Merav Parter. A polylogarithmic gossip algorithm for plurality consensus. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC'16*, pages 117–126, 2016.

[36] Tomoko Izumi, Keigo Kinpara, Taisuke Izumi, and Koichi Wada. Space-efficient self-stabilizing counting population protocols on mobile sensor networks. *Theor. Comput. Sci.*, 552:99–108, 2014.

[37] Adrian Kosowski and Przemyslaw Uznanski. Brief announcement: Population protocols are fast. In *Proceedings of the* 37*th ACM Symposium on Principles of Distributed Computing, PODC'18*, pages 475–477, 2018. Full version available at arXiv:1802.06872.

[38] George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *Proceedings of the* 41*st International Colloquium on Automata, Languages, and Programming, ICALP'14*, pages 871–882. Springer Berlin Heidelberg, 2014.

[39] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[40] Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In D. R. Avresky and Yann Busnel, editors, *Proceedings of the* 14*th IEEE International Symposium on Network Computing and Applications, NCA'15*, pages 35–42. IEEE Computer Society, 2015.

[41] Thomas Sauerwald and He Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *Proceedings of the* 53*rd Annual IEEE Symposium on Foundations of Computer Science, FOCS'12*, pages 341–350. IEEE Computer Society, 2012.

[42] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Brook. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 4(7):615–633, 2008.