A Population Protocol for Exact Majority with $O(\log^{5/3} n)$ Stabilization Time and $\Theta(\log n)$ States

Petra Berenbrink
Universität Hamburg, Hamburg, Germany
petra.berenbrink@uni-hamburg.de

Robert Elsässer[1]
University of Salzburg, Salzburg, Austria
elsa@cs.sbg.ac.at
https://orcid.org/0000-0002-5766-8103

Tom Friedetzky
Durham University, Durham, U.K.
tom.friedetzky@dur.ac.uk
https://orcid.org/0000-0002-1299-5514

Dominik Kaaser
Universität Hamburg, Hamburg, Germany
dominik.kaaser@uni-hamburg.de
https://orcid.org/0000-0002-2083-7145

Peter Kling
Universität Hamburg, Hamburg, Germany
peter.kling@uni-hamburg.de
https://orcid.org/0000-0003-0000-8689

Tomasz Radzik[2]
King's College London, London, U.K.
tomasz.radzik@kcl.ac.uk
https://orcid.org/0000-0002-7776-5461

Abstract

A population protocol is a sequence of pairwise interactions of n agents. During one interaction, two randomly selected agents update their states by applying a deterministic transition function. The goal is to stabilize the system at a desired output property. The main performance objectives in designing such protocols are small number of states per agent and fast stabilization time.

We present a fast population protocol for the exact-majority problem, which uses $\Theta(\log n)$ states (per agent) and stabilizes in $O(\log^{5/3} n)$ parallel time (i.e., in $O(n \log^{5/3} n)$ interactions) in expectation and with high probability. Alistarh et al. [SODA 2018] showed that exact-majority protocols which stabilize in expected $O(n^{1-\Omega(1)})$ parallel time and have the properties of monotonicity and output dominance require $\Omega(\log n)$ states. Note that the properties mentioned above are satisfied by all known population protocols for exact majority, including ours. They also showed an $O(\log^2 n)$-time exact-majority protocol with $O(\log n)$ states, which, prior to our work, was the fastest exact-majority protocol with polylogarithmic number of states. The standard design framework for majority protocols is based on $O(\log n)$ phases and requires that all

[1] Robert Elsässer's work has been supported by grant no. P 27613 of the Austrian Science Fund (FWF), "Distributed Voting in Large Networks".
[2] Tomasz Radzik's work has been supported by EPSRC grant EP/M005038/1, "Randomized algorithms for computer networks".

agents are well synchronized within each phase, leading naturally to upper bounds of the order of $\log^2 n$ because of $\Theta(\log n)$ synchronization time per phase. We show how this framework can be tightened with *weak synchronization* to break the $O(\log^2 n)$ upper bound of previous protocols.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Population Protocols, Randomized Algorithms, Majority

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.10

Related Version See [9], `https://arxiv.org/abs/1805.05157`, for the full version of this article.

1 Introduction

We consider population protocols [4] for exact-majority voting. The underlying computation system consists of a population of n anonymous (i.e., identical) *agents*, or *nodes*, and a *scheduler* which keeps selecting pairs of nodes for interaction. A *population protocol* specifies how two nodes update their states when they interact. The computation is a (perpetual) sequence of interactions between pairs of nodes. The objective is for the whole system to eventually stabilize in configurations which have the output property defined by the considered problem. In the general case, the nodes can be connected according to a specified graph $G = (V, E)$ and two nodes can interact only if they are joined by an edge. Following the scenario considered in most previous work on population protocols, we assume the complete communication graph and the random uniform scheduler. That is, each pair of (distinct) nodes has equal probability to be selected for interaction in any step and each selection is independent of the previous interactions.

The model of population protocols was proposed in Angluin et al. [4] and has subsequently been extensively studied to establish its computational power and to design efficient solutions for fundamental tasks in distributed computing such as various types of *consensus-reaching voting*. The survey from Aspnes and Ruppert [6] includes examples of population protocols, early computational results, and variants of the model. The main design objectives for population protocols are small number of states and fast stabilization time. The original definition of the model assumes that the agents are copies of the same finite-state automaton, so the number of states (per node) is constant. This requirement has later been relaxed by allowing the number of states to increase (slowly) with the population size, to study trade-offs between the memory requirements and the running times.

The (two-opinion) *exact-majority voting* is one of the basic settings of consensus voting [3, 4, 5]. Initially each node is in one of two distinct states q_A and q_B, which represent two distinct opinions (or votes) A and B, with a_0 nodes holding opinion A (starting in the state q_A) and b_0 nodes holding opinion B. We assume that $a_0 \neq b_0$ and denote the initial imbalance between the two opinions by $\epsilon = |a_0 - b_0|/n \geq 1/n$. The desired output property is that all nodes have the opinion of the initial majority. An *exact* majority protocol should guarantee that the correct answer is reached, even if the difference between a_0 and b_0 is only 1 (cf. [3]). In contrast, *approximate* majority would require correct answer only if the initial imbalance is sufficiently large. In this paper, when we refer to "majority" protocol/voting/problem we always mean the exact-majority notion.

Formal Model. We will now give further formalization of a population protocol and its time complexity. Let S denote the set of states, which can grow with the size n of the population (but keeping it low remains one of our objectives). A *configuration* of the system is an assignment of states to nodes. Let $q(v,t) \in S$ denote the state of a node $v \in V$ at step t (that is, after t individual interactions); $(v, q(v,t))_{v \in V}$ is the configuration of the system at this step. Two interacting nodes change their states according to a common deterministic transition function $\delta \colon S \times S \to S \times S$. A population protocol also has an *output function* $\gamma \colon S \to \Gamma$, which is used to specify the desired output property of the computation. For majority voting, $\gamma \colon S \to \{A, B\}$, which means that a node in a state $q \in S$ assumes that $\gamma(q)$ is the majority opinion. The system is in an (output) correct configuration at a step t if for each $v \in V$, $\gamma(q(v,t))$ is the initial majority opinion. We consider undirected individual communications, that is, the two interacting nodes are not designated as initiator and responder, so the transition functions must be symmetric. Thus if $\delta(q', q'') = (r', r'')$, then $\delta(q'', q') = (r'', r')$, implying, for example, that $\delta(q,q) = (r,r)$.

We say that the system is in a *stable configuration* if no node will ever again change its output in any configuration that can be reached. The computation continues (since it is perpetual) and nodes may continue updating their states, but if a node changes from a state q to another state q' then $\gamma(q') = \gamma(q)$. Thus a majority protocol is in a correct stable configuration if all nodes output the correct majority opinion and will do so in all possible subsequent configurations. Two main types of output guarantee categorize population protocols as either *always correct*, if they reach the correct stable configuration with probability 1, or *w.h.p. correct*. A protocol of the latter type reaches a correct stable configuration *w.h.p.*[3], allowing that with some low but positive probability an incorrect stable configuration is reached or the computation does not stabilize at all.

The notion of time complexity of population protocols which has lately been used to derive lower bounds on the number of states [1, 2], and the notion which we use also in this paper, is the *stabilization time* T_S defined as the first *round* when the system enters a correct stable configuration[4]. We follow the common convention of defining the *parallel time* as the number of interactions divided by n. Equivalently, we group the interactions in rounds of length n, called also (*parallel*) *steps*, and take the number of rounds as the measure of time. In our analysis we also use the term *period*, which we define as a sequence of n consecutive interactions, but not necessarily aligned with rounds.

The main result of this paper is a majority protocol with stabilization time $O(\log^{5/3} n)$ *w.h.p.* and in expectation while using logarithmically many states. According to [2] this number of states is asymptotically optimal for protocols with $\mathbf{E}(T_S) = O(n^{1-\epsilon})$, and to the best of our knowledge this is the first result that offers stabilization in time $O(\log^{2-\Omega(1)} n)$ with poly-logarithmic state space.

Related Literature. Draief and Vojnović [12] and Mertzios et al. [17] analyzed two similar four-state majority protocols. Both protocols are based on the idea that the two opinions have *weak* versions a and b in addition to the main *strong* versions A and B. The strong opinions are viewed as tokens moving around the graph. Initially each node v has a strong opinion A or B, and during the computation it has always one of the opinions a, b, A or B (so

[3] 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).

[4] Some previous papers (e.g. [1, 11]) refer to this stabilization time as the convergence time.

is in one of these four states). Two interacting opposite strong opinions cancel each other and change into weak opinions. Such pairwise canceling ensures that the difference between the numbers of strong opinions A and B does not change throughout the computation (remaining equal to $a_0 - b_0$) and eventually all strong opinions of the initial minority are canceled out. The surviving strong opinions keep moving around the graph, converting the weak opposite opinions. Eventually every node has the opinion (strong or weak) of the initial majority.

Mertzios et al. [17] called their protocol the *4-state ambassador protocol* (the strong opinions are ambassadors) and proved the expected stabilization time $O(n^5)$ for any graph and $O((n \log n)/|a_0 - b_0|)$ for the complete graph. Draief and Vojnović [12] called their 4-state protocol the *binary interval consensus*, viewing it as a special case of the *interval consensus* protocol of Bénézit et al. [7], and analyzed it in the continuous-time model. For the uniform edge rates (the continuous setting roughly equivalent to our setting of one random interaction per one time unit) they showed that the expected stabilization time for the complete graph is at most $2n(\log n + 1)/|a_0 - b_0|$. They also derived bounds on the expected stabilization time for cycles, stars and Erdős-Rényi graphs.

The appealing aspect of the four-state majority protocols is their simplicity and the constant-size local memory, but the downside is polynomially slow stabilization if the initial imbalance is small. The stabilization time decreases if the initial imbalance increases, so the performance would be improved if there were a way of boosting the initial imbalance. Alistarh et al. [3] achieved such boosting by multiplying all initial strong opinions by the integer parameter $r \geq 2$. The nodes keep the count of the number of strong opinions they currently hold. When eventually all strong opinions of the initial minority are canceled, $|a_0 - b_0|r$ strong opinions of the initial majority remain in the system. This speeds up both the canceling of strong opinions and the converting of weak opinions of the initial minority, but the price is the increased number of states. Refining this idea, Alistarh et al. [1] obtained a majority protocol which has stabilization time $O(\log^3 n)$ *w.h.p.* and in expectation and uses $O(\log^2 n)$ states.

A suite of constant-state polylogarithmic-time population protocols for various functions, including exact majority, was proposed by Angluin et al. [5]. Their protocols are *w.h.p.* correct and, more significantly, require a unique leader to synchronize the progress of the computation. Their majority protocol *w.h.p.* reaches a correct stable configuration within $O(\log^2 n)$ time (with the remaining low probability, it either needs more time to reach the correct output or it stabilizes with an incorrect output) and requires only a constant number of states, but the presence of the leader node is crucial.

The protocols developed in [5] introduced the idea of alternating *cancellations* and *duplications*, an idea that has been frequently used in subsequent majority protocols and also forms the basis of our new protocol. This idea has the following interpretation within the framework of canceling strong opinions. The canceling stops when it is guaranteed that *w.h.p.* the number of remaining strong opinions is less than δn, for some small constant $\delta < 1/2$. Now each remaining strong opinion duplicates (once): if a node with a strong opinion interacts with a node which does not hold a strong opinion then both nodes adopt the same strong opinion. This process of duplicating opinions lasts long enough to guarantee, again *w.h.p.*, that all strong opinions have been duplicated. One phase of (partial) cancellations followed by (complete) duplications takes *w.h.p.* $O(\log n)$ time, and $O(\log n)$ repetitions of this phase increases the difference between the numbers of strong opinions A and B to $\Theta(n)$. With such large imbalance between strong opinions, *w.h.p.* within additional $O(\log n)$ time the minority opinion is completely eliminated and the majority opinion is propagated to all nodes.

Bilke et al. [11] showed that the cancellation-duplication framework from [5] can be implemented without a leader if the agents have enough states to count their interactions. They obtained a majority protocol which has stabilization time $O(\log^2 n)$ *w.h.p.* and in expectation, and uses $O(\log^2 n)$ states. Berenbrink et al. [10] generalized the previous results on majority protocols by working with $k \geq 2$ opinions (*plurality voting*) and arbitrary graphs. Their protocol is based on the methodology introduced earlier for load balancing [18] and achieves $O(\operatorname{polylog} n)$ time using a polynomial number of states and assuming that the initial advantage of the most common opinion is $\Omega(n/\operatorname{polylog} n)$. For the case of complete graphs and $k = 2$, their protocol runs *w.h.p.* in $O(\log n)$ time.

Recently Alistarh et al. [2] showed that any majority protocol which has expected stabilization time of $O(n^{1-\epsilon})$, where ϵ is any positive constant, and satisfies the conditions of monotonicity and output dominance[5], requires $\Omega(\log n)$ states. They also presented a protocol which uses only $\Theta(\log n)$ states and has stabilization time $O(\log^2 n)$ *w.h.p.* and in expectation. Their lower and upper bounds raised the following questions. Can exact majority be computed in poly-logarithmic time with $o(\log n)$ states, if the time is measured in some natural and relevant way other than time until (correct) stabilization? Can exact majority be computed in $o(\log^2 n)$ time with poly-logarithmically many states? (The protocol in [2] and all earlier exact majority protocols which use poly-logarithmically many states have time complexity at least of the order of $\log^2 n$.) Regarding the first question, one may consider the *convergence time* instead of the stabilization time. For a random (infinite) sequence ω of interaction pairs, let $T_C = T_C(\omega)$ denote the convergence time, defined as the first round when (at some interaction during this round) the system enters a correct configuration (all nodes correctly output the majority opinion) and remains in correct configurations in all subsequent interactions (of this sequence ω). Clearly $T_C \leq T_S$, since reaching a correct stable configuration implies that whatever the future interactions may be, the system will always remain in correct configurations.

Very recently Kosowski and Uznański [16] and Berenbrink et al. [8] have shown that the convergence time T_C can be poly-logarithmic while using $o(\log n)$ states. In [16] the authors design a programming framework and accompanying compilation schemes that provide a simple way of achieving protocols (including majority) which are *w.h.p.* correct, use $O(1)$ states and converge in expected poly-logarithmic time. They can make their protocols always-correct at the expense of having to use $O(\log \log n)$ states per node, while keeping poly-logarithmic time, or increasing time to $O(n^{\epsilon})$, while keeping a constant bound on the number of states. In [8] the authors show 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 and 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 parameter $s \in [2, n]$.

The recent population protocols for majority voting often use similar technical tools (mainly efficient constructions of *phase clocks*) as protocols for another fundamental problem of *leader election*. There are, however, notable differences in computational difficulty of both problems, so advances in one problem do not readily imply progress with the other problem. For example, leader election admits always-correct protocols with poly-logarithmically fast stabilization and $\Theta(\log \log n)$ states [13] (the lower bound here is only $\Omega(\log \log n)$ [1]). Gasieniec and Stachowiak [14] have recently shown that leader election can be completed in

[5] 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.

expected time asymptotically significantly better than $\log^2 n$, but the best known time-bound for *w.h.p.*-correctness is $O(\log^2 n)$. The ideas in [14], however, are specific to leader election and we do not see how they could be applied to improve expected time of majority voting.

Our Contributions. We present a majority population protocol with stabilization time $O(\log^{5/3} n)$ *w.h.p.* and in expectation and $O(\log n)$ states. Since our protocol satisfies the conditions of monotonicity and output dominance, in view of the lower bound shown in [2], this implies the $O(\log n)$ number of states being asymptotically optimal for this type of protocols. The main contribution of our protocol is that no majority protocol with $O(\text{polylog}\, n)$ 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.

All known fast majority protocols with a poly-logarithmic number of states are based in some way on the idea (introduced in [5]) of a sequence of $\Omega(\log n)$ canceling-doubling phases. Each phase has length $\Omega(\log n)$ and the nodes are synchronized when they proceed from phase to phase. Our new protocol still uses the overall canceling-doubling framework (as explained in Section 2) but with *shorter phases* of length $\Theta(\log^{2/3} n)$ each, at the expense of weakening synchronization. We note that all existing majority protocols known to us cease to function properly with sub-logarithmic phases. Such phases are too short to synchronize nodes, resulting in there being, at the same time, nodes from different phases, and the computation potentially getting stuck (opposite opinions from different phases cannot cancel each other or we lose correctness). Moreover, we do not even have the guarantee that every node will be activated at all during a short phase – in fact, we know some nodes will not. The existing protocols require each node to be activated at least logarithmically many times during each phase.

Our main technical contributions are mechanisms to deal with the nodes which advance too slowly or too quickly through the short phases, that is, the nodes which are not in sync with the bulk. In a nutshell, we group $\log^{1/3} n$ phases in one *epoch*, show that the configuration of the system remains reasonably tidy throughout one epoch even without explicit synchronization, and introduce "cleaning-up" and synchronization at the boundaries between epochs. We believe that some of our algorithmic and analytical ideas developed for fast majority voting may be of independent interest.

Outline. The remainder of the paper is organized as follows. We first, in Section 2, describe the $O(\log^2 n)$-time, $O(\log^2 n)$-state `Majority` protocol presented in [11], which we use as the baseline implementation of the canceling-doubling framework. We refer to the structure and the main properties of this protocol when describing and analysing our new faster protocols. In Section 3 we present our main protocol `FastMajority1`, which stabilizes in $O(\log^{5/3} n)$ time and uses $\Theta(\log^2 n)$ states, and in Section 4 we outline the analysis of this protocol. In Section 5 we outline how to modify protocol `FastMajority1` yielding protocol `FastMajority2`, which has the same $O(\log^{5/3} n)$ bound on the running time but uses only $\Theta(\log n)$ states. Further details of our protocols, including pseudocode and detailed proofs, are given in the full version of the paper [9].

2 Exact majority: the general idea of canceling-doubling phases

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*. 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, the number of times it has undergone splitting. Thus the relation between the value z and the age g of a token is $z = 1/2^g$. Each node stores only the age of the token it possesses (if any), as its value can easily be computed from its age and vice-versa. Note that any sequence of canceling and splitting interactions preserves the difference between the sum 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. When only majority tokens are left in the system (and this is recognized by at least of the the nodes), the majority opinion can be propagated to all nodes *w.h.p.* within additional $O(\log n)$ time via a broadcast process. In the broadcast process, if two nodes interact and one of the nodes is in a final state, then the other node adopts the opinion of the first node and switches to the final state as well, except when a *conflict* occurs. In such a case some backup protocol is initiated that guarantees that the process always converges to the correct result. Since a conflict occurs with a small probability only, the running time of the overall protocol is $O(\log^2 n)$ with high probability and in expectation. The details of this standard process of propagating the outcome will be omitted from our descriptions and analysis. That is, from now on we assume that the objective is to eliminate the minority tokens.

From a node's local point of view, the computation of the $O(\log^2 n)$-time, $O(\log^2 n)$-state `Majority` protocol consists of at most $\log n + 2$ phases and each phase consists of at most $C \log n$ interactions, where C is a suitably large constant. Each node keeps count of phases and steps (interactions) within the current phase, and maintains further information which indicates the progress of computation. More precisely, each node v keeps the following data, which require $\Theta(\log^2 n)$ states.

- $v.token \in \{A, B, \emptyset\}$ – the type of token held by v. If $v.token = \emptyset$ then the node is empty.
- $v.phase \in \{0, 1, 2, \ldots, \log n + 2\}$ – the counter of phases.
- $v.phase\_step \in \{0, 1, 2, \ldots, (C \log n) - 1\}$ – the counter of steps in the current phase.
- Boolean flags, which are initially false and indicate the following status when set to *true*:
  - $v.doubled$ – v has a token which has already doubled in the current phase;
  - $v.done$ – the node has made the decision on the final output;
  - $v.fail$ – the protocol has failed because of some inconsistencies.

If a node v is in neither of the two special states *done* and *fail*, then we say that v is in a *normal state*: $v.normal \;\equiv\; \neg(v.done \vee v.fail)$. A node v is in Phase i if $v.phase = i$. If v is in Phase i and is not empty, then the age of the token at v is either i if $\neg v.doubled$ (the token has not doubled yet in this phase) or $i + 1$ if $v.doubled$. Thus the phase of a token (more correctly, the token's host node) and the flag *doubled* indicate the age of this token. Throughout the whole computation, the pair $(v.phase, v.phase\_step)$ can be regarded as the (combined) interaction counter $v.time \in \{0, 1, 2, \ldots, 2C \log^2 n)\}$ of node v. This counter is incremented by 1 at the end of each interaction. Thus, for example, if $v.phase\_step$ is equal to 0 after such an increment, then node v has just completed a phase. Each phase is divided into five parts defined below, where c is a constant discussed later.

- The beginning, the middle and the final parts of a phase are buffer zones, consisting of $c \log n$ steps each. The purpose of these parts is to ensure that the nodes progress through the current phase in a synchronized way.

- The second part is the *canceling stage* and the fourth part is the *doubling stage*, each consisting of $((C-3c)/2)\log n$ steps. If two interacting nodes are in the canceling stage of the same phase and have opposite tokens then the tokens are canceled. If two interacting nodes are in the doubling stage of the same phase, one of them has a token which has not doubled yet in this phase and the other is empty, then this is a doubling interaction.

If nodes were simply incrementing their step counters by 1 at each interaction, then those counters would start diverging too much for the canceling-doubling process to work correctly. An important aspect of the `Majority` protocol, as well as our new faster protocols, is the following mechanism for keeping the nodes sufficiently synchronized. When two interacting nodes are in different phases then the node in the lower phase jumps up to (that is, sets its step counter to) the beginning of the next phase. The `Majority` protocol relies on this synchronization mechanism in the high-probability case when all nodes are in two adjacent parts of a phase (that is, either in two adjacent parts of the same phase, or in the final part of one phase and the beginning part of the next phase.) In this case the process of pulling all nodes up to the next phase follows the pattern of *broadcast*. The node, or nodes, which have reached the beginning of the next phase by way of normal one-step increments broadcast the message "*if you are not yet in my phase then proceed to the next phase.*" By the time the broadcast is completed (that is, by the time when the message has reached all nodes), all nodes are together in the next phase. It can be shown that there is a constant β_0 such that *w.h.p.* the broadcast completes in $\beta_0 n\log n$ random pairwise interactions (see, for example [5]; other papers may refer to this process as *epidemic spreading* or *rumor spreading*).

The constant c in the definition of the parts of a phase is suitably smaller than the constant C, but sufficiently large to guarantee the following two conditions: (a) the *broadcast* from a given node to all other nodes completes *w.h.p.* within $(c/5)n\log n$ interactions; and (b) for a sequence of $(C/2)n\log n$ consecutive interactions, *w.h.p.* for each node v and each $0 < t \leq (C/2)n\log n$, the number of times v is selected for interaction within the first t interactions differs from the expectation $2t/n$ by at most $(c/5)\log n$. Condition (a) is used when the nodes reaching the end of the current phase i initiate broadcast to "pull up" the nodes lagging behind. Condition (a) implies that after $(c/5)n\log n$ interactions, *w.h.p.* all nodes are in the next phase. Using Condition (b) with $t = (c/5)n\log n$, we can also claim that *w.h.p.* at this point all nodes are within the first $(3/5)c\log n$ steps of the next phase (all nodes are in the next phase and no node interacted more than the expected $(2/5)c\log n$ plus $(1/5)c\log n$ times). Finally Condition (b) applied to all $(c/5)n\log n \leq t \leq (C/2)n\log n$ implies that *w.h.p.* the differences between the individual counts of node interactions do not diverge by more than $c\log n$ throughout this phase. We set $c = C^{3/4}$ and take C large enough so that $c \leq C/9$ (to have at least $3c\log n$ steps in the canceling and doubling stages) and both Conditions (a) and (b) hold. This way we achieve the following synchronized progress of nodes through a phase: *w.h.p.* all nodes are in the same part of the same phase before they start moving on to the next part. Moreover, also *w.h.p.*, for each canceling or doubling stage there is a sequence of $\Theta(n\log n)$ consecutive interactions when all nodes remain in this stage and each one of them is involved in at least $c\log n$ interactions.

Thus throughout the computation of the `Majority` protocol, *w.h.p.* all nodes are in two adjacent parts of a phase. In particular, *w.h.p.* the canceling and doubling activities of the nodes are separated. This separation ensures that the cancellation of tokens creates a sufficient number of empty nodes to accommodate new tokens generated by token splitting in the subsequent doubling stage. If two interacting nodes are not in the same or adjacent parts of a phase (a low, but positive, probability), then their local times (step counters) are considered inconsistent and both nodes enter the special *fail* state. The details of the `Majority` protocol are given in pseudocode in the full version [9].

From a global point of view, *w.h.p.* each new phase p starts with all nodes in normal states in the beginning of this phase. We say that this phase completes successfully if all nodes are in normal states in the beginning part of the next phase $p+1$. At this point all tokens have the same value $1/2^{p+1}$, and the difference between the numbers of opposite tokens is equal to $2^{p+1}|a_0 - b_0|$. The computation *w.h.p.* keeps successfully completing consecutive phases, each phase halving the value of tokens and doubling the difference between A tokens and B tokens, until the *critical phase* p_c, which is the first phase $0 \le p_c \le \log n - 1$ when the difference between the numbers of opposite tokens is

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

The significance of the critical phase is that the large difference between the numbers of opposite tokens means 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. Such a node enters the *done* state and broadcasts its (majority) opinion to all other nodes. In this case phase p_c is the *final phase*.

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, so the computation proceeds to the next phase $p_c + 1$. Phase $p_c + 1$ will be the final phase, since it will start with more than $(2/3)n$ tokens and all of them of the same type, so at least one token will fail to double and will assume that the computation has completed and will enter the *done* state. The failure to double is taken as an 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 (later receiving the message that the computation has completed) but *w.h.p.* no node reaches the end of phase $p_c + 2 \le \log n + 1$. Thus the *done* state is reached *w.h.p.* within $O(\log^2 n)$ parallel time.

The computation may fail *w.l.p.*[6] when the step counters of two interacting nodes are not consistent, or a node reaches phase $\log n + 2$, or two nodes enter the *done* state with opposite-type tokens. Whenever a node realizes that any of these low-probability events has occurred, it enters the *fail* state and broadcasts this state.

It is shown in [11] that the `Majority` protocol stabilizes, either in the correct all-*done* configuration or in the all-*fail* configuration, within $O(\log^2 n)$ time *w.h.p.* and in expectation. The standard technique of combining a fast protocol, which *w.l.p.* may fail, with a slow always-correct backup protocol gives an *extended* `Majority` protocol, which requires $\Theta(\log^2 n)$ states per node and computes the exact majority within $O(\log^2 n)$ time *w.h.p.* and in expectation. The idea 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, 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.

We omit the details of using a slow backup protocol (see, for example, [2, 11]), and assume that the objective of a canceling-doubling protocol is to use a small number of states s, to compute the majority quickly *w.h.p.*, say within a time bound $T'(n)$, and to also have low expected time of reaching the correct all-*done* configuration or the all-*fail* configuration, say within a bound $T''(n)$. If the bounds $T'(n)$ and $T''(n)$ are of the same order $O(T(n))$, then we get a corollary that the majority can be computed with $O(s)$ states in $O(T(n))$ time *w.h.p.* and in expectation.

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

3 Exact majority in $O(\log^{5/3} n)$ time with $\Theta(\log^2 n)$ states

To improve on the $O(\log^2 n)$ time of the `Majority` protocol, we reduce the length of a phase to $\Theta(\log^{1-a} n)$, where $a = 1/3$. The new `FastMajority1` protocol runs in $O(\log^{1-a} n) \times O(\log n) = O(\log^{5/3} n)$ time and requires $\Theta(\log^2 n)$ states per node. We will show in Section 5 that the number of states can be reduced to asymptotically optimal $\Theta(\log n)$. We keep the term a in the description and the analysis of our fast majority protocols to simplify notation and to make it easier to trace where a larger value of a would break the proofs.

Phases of sub-logarithmic length are too short to ensure that *w.h.p.* all tokens progress through the phases synchronously and keep up with required canceling and doubling, as they did in the `Majority` protocol. In the `FastMajority1` 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 phase. Such tokens stop contributing to the regular dynamics of canceling and doubling. The general idea of our solution is to group $\log^a n$ consecutive phases (a total of $\Theta(\log n)$ steps) into an *epoch*, to attach further $\Theta(\log n)$ steps at the end of each epoch to enable the out-of-sync tokens to reach the age required at the end of this epoch, and to synchronize all nodes by the broadcast process at the boundaries of epochs. When analyzing the progress of tokens through the phases of the same epoch, we consider the tokens which remain synchronized and the out-of-sync tokens separately.

We now proceed to the details of the `FastMajority1` protocol. Each epoch consists of $2C \log n$ steps, where C is a suitably large constant, and is divided into two equal-length parts. The first part is a sequence of $\log^a n$ canceling-doubling phases, each of length $C \log^{1-a} n$. The purpose of the second part is to give sufficient time to out-of-sync tokens so that *w.h.p.* they all complete all splitting required for this epoch. Each node v maintains the following data, which can be stored using $\Theta(\log^2 n)$ states. For simplicity of notation, we assume that expressions like $\log^a n$ and $C \log^{1-a} n$ have integer values if they refer to an index (or a number) of phases or steps.

- $v.token \in \{A, B, \emptyset\}$ – type of token held by v.
- $v.epoch \in \{0, 1, \ldots, \log^{1-a} n + 2\}$ - the counter of epochs.
- $v.age\_in\_epoch \in \{0, 1, \ldots, \log^a n\}$ – the age of the token at v (if v has a token) with respect to the beginning of the current epoch. If $v.token$ is A or B, then the age of this token is $g = v.epoch \cdot \log^a n + v.age\_in\_epoch$ and the value of this token is $1/2^g$.
- $v.epoch\_part \in \{0, 1\}$ – each epoch consists of two parts, each part has $C \log n$ steps. The first part, when $v.epoch\_part = 0$, is divided into $\log^a n$ canceling-doubling phases.
- $v.phase \in \{0, 1, \ldots, (\log^a n) - 1\}$ – counter of phases in the first part of the current epoch.
- $v.phase\_step \in \{0, 1, \ldots, (C \log^{1-a} n) - 1\}$ – counter of steps (interactions) in the current phase.
- Boolean flags indicating the status of the node, all set initially to *false*:
  - $v.doubled$, $v.done$, $v.fail$ – as in the `Majority` protocol;
  - $v.out\_of\_sync$ – v has a token which no longer follows the expected progress through the phases of the current epoch;
  - $v.additional\_epoch$ – the computation is in the additional epoch of $3 \log^a n$ phases, with each of these phases consisting now of $\Theta(\log n)$ steps.

We say that a node v is in epoch j if $v.epoch = j$, and in phase i (of the current epoch) if $v.phase = i$. We view the triplet $(v.epoch\_part, v.phase, v.phase\_step)$ as the (combined) counter $v.epoch\_step \in \{0, 1, 2, \ldots, (2C \log n) - 1\}$ of steps in the current epoch, and the pair $(v.\mathit{epoch}, v.\mathit{epoch\_step})$ as the counter $v.\mathit{time} \in \{0, 1, 2, \ldots, (2C\log^{2-a} n) + O(\log n)\}$ of the steps of the whole protocol. If a node v is not in any of the special states *out_of_sync*, *additional_epoch*, *done* or *fail*, then we say that v is in a *normal state*:

$$v.\mathit{normal} \;\equiv\; \neg(v.\mathit{out\_of\_sync} \vee v.\mathit{additional\_epoch} \vee v.\mathit{done} \vee v.\mathit{fail}).$$

A normal token is a token hosted by a normal node. Each phase is split evenly into the canceling stage (the first $(C/2)\log^{1-a} n$ steps of the phase) and the doubling stage (the remaining $(C/2)\log^{1-a} n$ steps).

The vast majority of the tokens are normal tokens progressing through the phases of the current epoch in a synchronized fashion. These tokens are simultaneously in the beginning part of the same phase j and have the same age j (w.r.t. the end of the epoch). They first try to cancel with tokens of the same age but opposite type during the canceling stage, and if they survive, they then split during the subsequent doubling stage. At some later time most of the tokens will still be normal, but in the beginning part of the next phase $j+1$ and having age $j+1$. Thus the age of a normal token (w.r.t. the beginning of the current epoch) is equal to its phase if the token has not yet split in this phase (this is recorded by setting the flag *doubled*), or to its phase plus 1 otherwise.

As in the `Majority` protocol, we separate the canceling and the doubling activities to ensure that the canceling of tokens first creates a sufficient number of empty nodes to accommodate the new tokens obtained later from splitting. Unlike in the `Majority` protocol, the `FastMajority1` protocol does not have the buffer zones within a phase. Such zones would not be helpful in the context of shorter sublogarithmic phases when anyway we cannot guarantee that *w.h.p. all* nodes progress through a phase in a synchronously.

A token which has failed to split in one of the phases of the current epoch becomes an out-of-sync token (the *out_of_sync* flag is set). Such a token no longer follows the regular canceling-doubling phases of the epoch, but instead tries cascading splitting to break up into tokens of age $\log^a n$ (relative to the beginning of the epoch) as expected by the end of this epoch. An out-of-sync token does not attempt canceling because there would be only relatively few opposite tokens of the same value, so only a small chance to meet them (too small to make a difference in the analysis). The tokens obtained from splitting out-of-sync tokens inherit the out-of-sync status. A token drops the out-of-sync status if it is in the second part of the epoch and has reached the age $\log^a n$. (Alternatively, out-of-sync tokens could switch back to the normal status as soon as their age coincides again with their phase, but this would complicate the analysis.) An *out-of-sync node* is a node hosting an out-of-sync token. While each normal node and token is in a specific phase of the first part of an epoch or is in the second part of an epoch, the out-of-sync nodes (tokens) belong to an epoch but not to any specific phase. The objective for a normal token is to split into two halves in each phase of the current epoch (if it survives canceling). The objective of an out-of-sync token is to keep splitting in the current epoch (disregarding the boundaries of phases) until it breaks into tokens as expected by the end of this epoch.

In our analysis we show that *w.h.p.* there are only $O(n/2^{\Theta(\log^a n)})$ out-of-sync tokens in any one epoch. *W.h.p.* all out-of-sync tokens in the current epoch reach the age $\log^a n$ (w.r.t. the beginning of the epoch) by the midpoint of the second part of the epoch (that is, by the step $(3/2)C\log n$ of the epoch), for each but the *final epoch* j_f. 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 `Majority` protocol where on completion of the final phase *w.h.p.* only majority tokens are left, in the `FastMajority1` 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, identifying that way that this is the final epoch, enters the *additional_epoch* state and broadcasts this state through the system to trigger an *additional epoch* of $\Theta(\log^a n)$ phases. More precisely, the additional epoch consists of at most $3\log^a n$ phases corresponding to epochs $j_f - 1$ (if $j_f > 0$), j_f and $j_f + 1$, each phase now having $\Theta(\log n)$ steps. *W.h.p.* these phases include the critical phase p_c and the phase $p_c + 1$, defined by (1). The computation of the additional epoch is as per the `Majority` protocol, taking $O(\log^{1+a} n)$ time to reach the correct all-*done* configuration *w.h.p.* or the all-*fail* configuration *w.l.p.*

Two interacting nodes first check the consistency of their *time* counters (the counters of interactions) and switch to *fail* states if the difference between the counters is greater than $(1/4)C\log n$. If the counters are consistent but the nodes are in different epochs (so one near the end of an epoch with the other being near the beginning of the next) then the node in the lower epoch jumps up to the beginning of the next epoch. This is the synchronization mechanism at the boundaries of epochs, analogous to the synchronization by broadcast at the boundaries of phases in the `Majority` protocol. In the `FastMajority1` protocol, however, it is not possible to synchronize the nodes at the boundaries of (short) phases.

For details of the `FastMajority1` protocol we refer the reader to the full version [9].

4 Analysis of the FastMajority1 protocol

Ideally we would like for all tokens to progress through the phases of the current epoch synchronously, *w.h.p.*, that is, all tokens being roughly in the same part of the same phase, as in the `Majority` protocol. This would mean that *w.h.p.* at some (global) time all nodes are in the beginning part of the same phase, ensuring that all tokens have the same value x, and at some later point all nodes are in the end part of this phase and all surviving tokens have value $x/2$. This ideal behavior is achieved by the `Majority` protocol at the cost of having $\Theta(\log n)$-step phases. As discussed in Section 2, the logarithmic length of a phase also gives sufficient time to synchronize *w.h.p.* the local times of all nodes at the end of a phase so that they all end up together in the beginning part of the next phase.

Now, with phases having only $\Theta(\log^{1-a} n)$ steps, we face the following two difficulties in the analysis. Firstly, while a good number of tokens split during such a shorter phase, *w.h.p.* there are also some tokens which do not split. Secondly, phases of length $o(\log n)$ are too short to keep the local times of the nodes synchronized. We can again show that a good number of nodes proceed in synchronously, but *w.h.p.* there are nodes falling behind or rushing ahead and our analysis has to account for them.

Counting the phases across the epochs, we define the critical phase p_c as in (1). Similarly as in the $O(\log^2 n)$-time `Majority` protocol, the computation proceeds through the phases moving from epoch to epoch until it reaches the critical phase p_c. The computation will then get stuck in this phase or in the next phase $p_c + 1$. Some tokens do not split in that final phase, nor in any subsequent phase of the current epoch, because there are not enough empty nodes to accommodate new tokens. Almost all minority tokens have been eliminated, and so the creation of empty nodes by cancellations of opposite tokens has all but stopped. This is the final epoch j_f and the nodes which do not split to the value required by the end of this epoch trigger the additional epoch of $O(\log^a n)$ phases, each having $\Theta(\log n)$ steps. The additional epoch is needed because we do not have a high-probability guarantee that all minority tokens are eliminated by the end of the final epoch. The small number of remaining minority tokens may have various values which are inconsistent with the values of the majority tokens, so further cancellations of tokens might not be possible. The additional epoch includes the phases of the three consecutive epochs $j_f - 1, j_f$ and $j_f + 1$ to ensure that *w.h.p.* both phases p_c and $p_c + 1$ are included. Phase p_c can be as early as the last phase in epoch $j_f - 1$ and phase $p_c + 1$ can be as late as the first phase in epoch $j_f + 1$.

The following conditions describe the regular configuration of the whole system at the beginning of epoch j, and the corresponding Lemma 1 summarizes the progress of the computation through this epoch. Recall that the `FastMajority1` protocol is parameterized by a suitably large constant $C > 1$ and our analysis refers also to another smaller constant $c = C^{3/4}$. We refer to the first (last) $c \log^{1-a} n$ steps of a phase or a stage as the beginning (end) part of this phase or stage. The (global) time steps count the number of interactions of the whole system.

EpochInvariant(j) :

1. At least $n(1 - 1/2^{3 \log^a n})$ nodes are in normal states, are in epoch j, and their *epoch_step* counters are at most $c \log^a n$.
2. For each remaining node u,
   a. u is in a normal state in epoch $j - 1$ and $u.\mathit{epoch\_step} \geq (3/2)C \log n$ (that is, u is in the last quarter of epoch $j - 1$), or
   b. u is in a normal or out-of-sync state in epoch j and $u.\mathit{epoch\_step} \leq 4c \log n$.

▶ **Lemma 1.** *Consider an arbitrary epoch $j \geq 0$ such that phase p_c belongs to an epoch $j' \geq j$ and assume that at some (global) step t the condition EpochInvariant(j) holds.*

1. *If phase p_c does not belong to epoch j (that is, phase p_c is in a later epoch $j' > j$), then w.h.p. there is a step $\tilde{t} \leq t + 2Cn \log n$ when the condition EpochInvariant$(j + 1)$ holds.*
2. *If both phases p_c and $p_c + 1$ belong to epoch j, then w.h.p. there is a step $\tilde{t} \leq t + 2Cn \log n$ when*

   $(*)$ *a node completes epoch j and enters the additional_epoch state (because it has a token which has not split to the value required by the end of this epoch); and all other nodes are in normal or out-of-sync states in the second part of epoch j or the first part of epoch $j + 1$.*
3. *Otherwise, that is, if phase p_c is the last phase in epoch j (and $p_c + 1$ is the first phase in epoch $j + 1$), then w.h.p. either there is a step $\tilde{t} \leq t + 2Cn \log n$ when the above condition $(*)$ for the end of epoch j holds, or all nodes eventually proceed to epoch $j + 1$ and there is a step $\hat{t} \leq t + 3Cn \log n$ when the condition analogous to $(*)$ but for the end of epoch $j + 1$ holds.*

The condition *PhaseInvariant1*(j, i) given below describes the regular configuration of the whole system at the beginning of phase $0 \leq i \leq \log^a n$ in epoch $j \geq 0$. We note that the last phase in an epoch is phase $\log^a n - 1$ and the condition *PhaseInvariant1*$(j, \log^a n)$ refers in fact to the beginning of the second part of the epoch. A normal token in the beginning of phase i in epoch j has (absolute) value $2^{-(j \log^a n + i)}$ and relative values 1, 2, $1/2^i$ and $2^{\log^a n - i}$ w.r.t. the beginning of this phase, the end of this phase, the beginning of this epoch and the end of this epoch, respectively. It may also be helpful to recall that for a given node v, phase i starts at v's epoch step $Ci \log^{1-a} n$. Observe that *EpochInvariant*(j) implies *PhaseInvariant1*$(j, 0)$.

PhaseInvariant1(j, i) :

1. The set W of nodes which are normal and in the beginning part of phase i in epoch j has size at least $n(1 - (i+1)/2^{2 \log^a n})$. That is, a node v is in W if and only if $v.\mathit{normal}$ is true, $v.\mathit{phase\_step} \leq c \log^{1-a} n$, $v.\mathit{epoch} = j$, and either $v.\mathit{epoch\_part} = 0$ and $v.\mathit{phase} = i$ if $i < \log^a n$, or $v.\mathit{epoch\_part} = 1$ and $v.\mathit{phase} = 0$ if $i = \log^a n$.

2. Let $U = V \setminus W$ denote the set of the remaining nodes.
   a. For each $u \in U$:
      u is a normal node in epoch $j-1$, $u.\mathit{epoch\_step} \geq (3/2)C \log n$ and $i < (c/C)\log^a n$; or u is in a normal or out-of-sync state in epoch j and $|u.\mathit{epoch\_step} - Ci\log^{1-a} n| \leq 4c\log n$.
   b. The total value of the tokens in U w.r.t. the end of epoch j is at most $n(i+1)/2^{2\log^a n}$.

For an epoch $0 \leq j$ and a phase $0 \leq i < \log^a n$ in this epoch, let $p(j,i) = j\log^a n + i$ denote the global index of this phase. We show that *w.h.p.* the condition *PhaseInvariant1*(j,i) holds at the beginning of each phase $p(j,i) \leq p_c$.

▶ **Lemma 2.** *For arbitrary $0 \leq j$ and $0 \leq i \leq \log^a n - 1$ such that $p(j,i) \leq p_c$, assume that the condition EpochInvariant(j) holds at some (global) time step t and the condition PhaseInvariant1(j,i) holds at the step $t_i = t + i(C/2)n\log^{1-a} n$. Then the following conditions hold, where $t_{i+1} = t + (i+1)(C/2)n\log^{1-a} n$.*
1. *If $p(j,i) < p_c$, then w.h.p. at step t_{i+1} the condition PhaseInvariant1$(j,i+1)$ holds.*
2. *If $p(j,i) = p_c$, then w.h.p. at step t_{i+1} the total value, w.r.t. the end of epoch j, of the minority-opinion tokens is $O(n\log n/2^{2\log^a n})$.*

Lemma 2 is proven by analyzing the cancellations and duplications of tokens in one phase. This lemma heavily uses Claim 6, in which it is essential that $a \leq 1/3$. Lemma 1 is proven by inductively applying Lemma 2. In turn, Theorem 3 below, which states the $O(\log^{5/3} n)$ bound on the completion time of the `FastMajority1` protocol, can be proven by inductively applying Lemma 1 and by choosing $a = 1/3$.

▶ **Theorem 3.** *The `FastMajority1` protocol uses $\Theta(\log^2 n)$ states, computes the majority w.h.p. within $O(\log^{5/3} n)$ time, and reaches the correct all-done configuration or the all-fail configuration within expected $O(\log^{5/3} n)$ time.*

▶ **Corollary 4.** *The majority can be computed with $\Theta(\log^2 n)$ states in $O(\log^{5/3} n)$ time w.h.p. and in expectation.*

We now give some further explanation of the structure of our analysis, referring the reader to the full version [9] for the formal proofs. Lemma 5 and Claim 6 show the synchronization of the nodes which we rely on in our analysis. Lemma 5 is used in the proof of Lemma 1, where we analyze the progress of the computation through one epoch consisting of $O(n\log n)$ interactions ($O(\log n)$ parallel steps). Lemma 5 can be easily proven using first Chernoff bounds for a single node and then the union bound over all nodes. The proof of Claim 6 is considerably more involved, but we need this claim in the proof of Lemma 2, where we look at the finer scale of individual phases and have to consider intervals of $\Theta(\log^{1-a} n)$ interactions of a given node. This claim shows, in essence, that most of the nodes stay tightly synchronized when they move from phase to phase through one epoch. The *epoch_step* counters of these nodes stay in a range of size at most $c\log^{1-a} n$.

▶ **Lemma 5.** *For any constant C and for $c = C^{3/4}$, during a sequence of $t \leq 2Cn\log n$ interactions, with probability at least $1 - n^{-\alpha(C)}$ (for a suitable function $\alpha = \omega(1)$) the number of interactions of each node is within $c\log n$ of the expectation of $2t/n$ interactions.*

▶ **Claim 6.** *For a fixed $j \geq 0$, assume that EpochInvariant(j) holds at a time step t. Let $W \subseteq V$ be the set of $n(1-o(1))$ nodes which satisfy at this step the condition 1 of EpochInvariant(j) (that is, W is the set of nodes which are in epoch j with epoch_step counters at most $c\log^a n$). Then at an arbitrary but fixed time step $t < t' \leq t + (3/4)Cn\log n$,* *w.h.p. all nodes in W are in epoch j and all but $O(n/2^{6\log^a n})$ of them have their epoch_step counters within $c/2 \cdot \log^{1-a} n$ from $2(t'-t)/n$.*

Note that this claim only holds if $a \le 1/3$, otherwise one can not guarantee that *w.h.p.* all but $O(n/2^{6\log^a n})$ of the nodes in W have their *epoch_step* counters within $c/2 \cdot \log^{1-a} n$ of $2(t'-t)/n$.

Lemmas 7 and 8 describe the performance of the broadcast process in the population-protocol model. Lemma 7 has been used before and is proven, for example, in [11]. Lemma 8 is a more detailed view at the dynamics of the broadcast process, which we need in the context of Lemma 1 to show that the synchronization at the end epoch j gives *w.h.p.* *EpochInvariant*$(j+1)$.

▶ **Lemma 7.** *For any constant c, the broadcast completes with probability at least $1 - n^{-\alpha(c)}$ (for a suitable function $\alpha = \omega(1)$) within $cn\log n$ interactions.*

▶ **Lemma 8.** *Let $b \in (0,1)$ and $c > 0$ be arbitrary constants and let c_1 be a sufficiently large constant. Consider the broadcast process and let t_1 be the first step when $n/2^{6\log^b n}$ nodes are already informed and $t_2 = t_1 + c_1 n \log^b n$. Then the following conditions hold.*

1. *With probability at least $1 - n^{-\omega(1)}$, $n - O(n/2^{6\log^b n})$ nodes receive the message for the first time within the $c_1 n \log^b n$ consecutive interactions $\{t_1+1, t_1+2, \ldots, t_2\}$.*
2. *With probability at least $1 - n^{-\alpha(c)}$ (for a suitable function $\alpha = \omega(1)$), $t_1 \le cn\log n$ and each node interacts at most $4c\log n$ times in interval $[1, t_2]$.*
3. *With probability at least $1 - n^{-\omega(1)}$, there are $n - O(n/2^{6\log^b n})$ nodes which interact within interval $[t_1+1, t_2]$ at least $c_1 \log^b n$ times but not more than $3c_1 \log^b n$ times.*

5 Reducing the number of states to $\Theta(\log n)$

Our `FastMajority1` protocol described in Section 3 requires $\Theta(\log^2 n)$ states per node. Using the idea underlying the constructions of *leaderless phase clocks* in [15] and [2], we now modify `FastMajority1` into the protocol `FastMajority2`, which still works in $O(\log^{5/3} n)$ time but has only the asymptotically optimal $\Theta(\log n)$ states per node.[7] The general idea is to separate from the whole population a subset of *clock nodes*, whose only functionality is to keep the time for the whole system. The other nodes work on computing the desired output and check whether they should proceed to the next stage of the computation when they interact with clock nodes. We note that while we use similar general structure and terminology as in [2], the meaning of some terms and the dynamics of our phase clock are somewhat different. A notable difference 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, we keep the updates of time counters as in the `Majority` and `FastMajority1` protocols: 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 phase or epoch, if the faster node is already there.

The nodes in the `FastMajority2` protocol are partitioned into two sets with $\Theta(n)$ nodes in each set. One set consists of *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

[7] It may be possible to use instead the ideas underlying other phase clocks, e.g. the $\Theta(\log\log n)$-state phase clock from [13], but this would not result in fewer states being needed for our protocol.

(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 stage of a phase), but it does not maintain a detailed step counter. The other set consists of *clock nodes*, which maintain their detailed epoch-step counters, counting 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 epoch. Thus the clock nodes update their counters in the same way as all nodes would update their counters in the `FastMajority1` protocol.

The worker nodes interact with each other in a similar way as in `FastMajority1`, 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 will 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 in protocols `Majority` and `FastMajority1`.

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 pre-processing. 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 toke 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 pre-processing, but the $O(\log n)$ states needed for this are re-used when the pre-processing 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 (one may view them as if already split in the first phase) and some nodes may be empty.

Referring to the state space of the `FastMajority1` protocol, in the `FastMajority2` protocol each worker node v maintains data fields $v.token$, $v.epoch$ and $v.age\_in\_epoch$ to carry information about tokens and their ages, and a constant number of flags to keep track of the status of the node and its progress through the current epoch and the current phase. These include the status flags from the `FastMajority1` protocol $v.doubled$, $v.out\_of\_sync$ and $v.additional\_epoch$, and flags indicating the progress: the $v.epoch\_part$ flag from `FastMajority1` and a new (multi-valued) flag $stage \in \{beginning, canceling, middle, doubling, ending\}$. The clock nodes maintain the *epoch_step* counters. The nodes have constant number of further flags, for example to support the initialization to workers and clocks and the implementation of the additional epoch and the slow backup protocol. Thus in total each node has only $\Theta(\log n)$ states.

Further details of `FastMajority2`, including pseudocodes, details of the pre-processing and outline of the proof of Theorem 9 which summarizes the performance of this protocol, are given in the full version [9].

▶ **Theorem 9.** *The* `FastMajority2` *protocol uses* $\Theta(\log n)$ *states, computes the exact majority w.h.p. within* $O(\log^{5/3} n)$ *parallel time and stabilizes (in the correct all-done configuration or in the all-fail configuration) within the expected* $O(\log^{5/3} n)$ *parallel time.*

▶ **Corollary 10.** *The exact majority can be computed with* $\Theta(\log n)$ *states in* $O(\log^{5/3} n)$ *parallel time w.h.p. and in expectation.*

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

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 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2221–2239, 2018. doi:10.1137/1.9781611975031.144.

3 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 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 47–56. ACM, 2015. URL: http://dl.acm.org/citation.cfm?id=2767386, doi:10.1145/2767386.2767429.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006. doi:10.1007/s00446-005-0138-3.

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

6 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.

7 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 2009, 19-24 April 2009, Taipei, Taiwan*, pages 3661–3664. IEEE, 2009. doi:10.1109/ICASSP.2009.4960420.

8 Petra Berenbrink, Robert Elässser, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Majority & stabilization in population protocols. Unpublished manuscript, available on arXiv, May 2018.

9 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with $O(\log^{5/3} n)$ stabilization time and asymptotically optimal number of states. Unpublished manuscript, available on arXiv, May 2018. arXiv:1805.05157.

10 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. URL: http://arxiv.org/abs/1602.01342.

11 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 Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 451–453. ACM, 2017. Full version available at arXiv:1705.01146. doi:10.1145/3087801.3087858.

12 Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. In *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*, pages 1792–1800. IEEE, 2010. doi:10.1109/INFCOM.2010.5461999.

13 Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete*

Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, pages 2653–2667, 2018. doi:10.1137/1.9781611975031.169.

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

15 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*, pages 117–126, 2016.

16 A. Kosowski and P. Uznański. Population Protocols Are Fast. *ArXiv e-prints*, 2018. arXiv:1802.06872v2.

17 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 Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, volume 8572 of *Lecture Notes in Computer Science*, pages 871–882. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-43948-7_72.

18 Thomas Sauerwald and He Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 341–350. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.86.