On Counting the Population Size

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

Dominik Kaaser
Universität Hamburg, Hamburg, Germany
dominik.kaaser@uni-hamburg.de

Tomasz Radzik
King’s College London, London, UK
tomasz.radzik@kcl.ac.uk

June 1, 2019

Abstract
We consider the problem of counting the population size in the population model. In this model, we are given a distributed system of n identical agents which interact in pairs with the goal to solve a common task. In each time step, the two interacting agents are selected uniformly at random. In this paper, we consider so-called uniform protocols, where the actions of two agents upon an interaction may not depend on the population size n. We present two population protocols to count the size of the population: protocol APPROXIMATE, which computes with high probability either ⌊log n⌋ or ⌈log n⌉, and protocol COUNTEXACT, which computes the exact population size in optimal O(n log n) interactions, using O(n) states. Both protocols can also be converted to stable protocols that give a correct result with probability 1 by using an additional multiplicative factor of O(log n) states.

1 Introduction
In this paper we consider the problem of counting the population size in the probabilistic population model. The model was introduced in [5] to model distributed systems of resource-limited mobile agents, which interact with each other in order to solve a common task. The computation of a probabilistic population protocol can be viewed as a sequence of pairwise interactions of randomly chosen agents. In each interaction, the two participating agents observe each others’ states and update their own state according to a transition function common to all agents.

In this setting, we are interested in so-called uniform population protocols, where the transition function does not depend on the size of the population, so it can be applied to any population size. In the original definition of population protocols in [5], each agent is a copy of the same finite state machine. Such a protocol is by definition uniform, as its state space has constant size (independent of the size of the population). However, more recent results have shown that many problems can be solved much faster if agents are equipped
with a number of states that grows with the population size. For example, a simple protocol with a constant number of states solves the majority problem in expected $O(n^2)$ interactions [17, 20]. On the other hand, a number of protocols have been proposed which solve the majority problem in $O(n \text{polylog } n)$ interactions using $O(\text{polylog } n)$ states [1, 2, 9, 12]. These protocols are all non-uniform since their transition functions refer to values which have to be $\Theta(\log n)$. We note that $\Omega(n \log n)$ interactions are required to reach a positive constant probability that each agent participates in at least one interaction, so $\Omega(n \log n)$ is a lower bound on the number of interactions of a protocol which solves any nontrivial problem.

Turning to the problem of counting the population size, either exactly or approximately, we naturally have to consider population protocol models with growing memory, where lower bounds on the required number of states depend on the desired counting accuracy. The exact counting of the population size requires $\Omega(n)$ states, while estimating the population size up to a constant factor requires $\Omega(\log n)$ states.

There is a simple and uniform protocol for exact population counting, which completes in expected $\Theta(n^2)$ interactions and uses $\Theta(n^2)$ states: the agents start with one token each and keep combining the tokens into bags, propagating at the same time the maximum size of a bag and using that maximum as their current output. The transition function of this protocol does not refer in any way to (any estimate of) the size of the population.

In this paper we are interested in uniform population counting protocols which run in $O(n \text{polylog } n)$ interactions and use a small number of states (relatively to the lower bounds). We present new protocols for exact and approximate population counting, which use a significantly smaller number of states than the previously best protocols shown in [13, 14]. Our protocols have also further desirable properties. Our exact protocol is the first protocol for this problem which completes the computation within asymptotically optimal $O(n \log n)$ interactions. A variant of our approximate protocol is the first $O(\text{polylog } n)$-state, $O(n \text{polylog } n)$-interaction protocol always converging to a value which is within a constant factor from the exact size of the population. To give formal statements of the previous results and our new results, we have to introduce first some details of the model and the way the efficiency of population protocols is measures.

### 1.1 Computation Model

The computation model is a population of $n$ agents which are capable of performing local computations. A population protocol is specified by a state space $Q$, an output domain $O$, a transition function $\delta : Q \times Q \rightarrow Q \times Q$, and an output function $\omega : Q \rightarrow O$. Each agent has a state $q \in Q$, which is updated during interactions. The current output of an agent in state $q$ is $\omega(q)$. The current configuration of the system is the vector from $Q^n$ with the current states of the agents. The computation of a population protocol is a sequence of pairwise interactions of agents. In every time step, a probabilistic scheduler selects independently and uniformly at random a pair of agents for interaction, with the first agent called the initiator and the second the responder. During the interaction the agents update their states by applying the transition function $\delta$. Such an update is denoted by $(x, y) \rightarrow (x', y')$, where $(x, y)$ refers to the states of the agents before the interaction and $(x', y') = \delta(x, y)$ to the states after the interaction.

A given problem which we want to solve by population protocols specifies the set of initial (input) configurations, the output domain $O$, and the desired (output) configurations for given input configurations. For the exact population counting problem, all agents are initially in the same state $q_0$ and the output domain is the set of positive integers. A desired configuration is when all agents output correctly the (exact) size of the population. In the leader election problem, which arises as a sub-problem in many population protocols, including in the protocols for the population counting problem, all agents start with the same initial state and the output domain is $O = \{\text{leader, follower}\}$. A desired configuration is when exactly one agent outputs that it is the leader.

The following two definitions are commonly used to capture important aspects of the time efficiency of population protocols. The convergence time $T_C$ of an execution of a protocol is the number of interactions until the system enters a desired configuration and never leaves the set of desired configurations again. The stabilization time $T_S$ of an execution of a protocol is the number of interactions until the system enters a desired stable configuration, meaning that starting from this configuration no sequence of pairwise interactions can take the system outside of the set of desired configurations. We always have $T_C \leq T_S$ for any execution of a protocol, but the stabilization time may be strictly greater than the convergence time. A population protocol is always correct (or stable), if for each initial configuration the computation reaches a correct stable
configuration with probability 1, or \( w.h.p. \) correct, if there is a small (of the order of \( n^{-\Omega(1)} \)), but positive, probability that the system settles with an incorrect output or does not settle at all. We say that a protocol converges (resp. stabilizes) in \( T(n) \) time \( w.h.p. \) (resp. in expectation), if \( T_C \) (resp. \( T_S \)) is at most \( T(n) \) \( w.h.p. \) (resp. in expectation).\(^1\)

The second measure of efficiency of population protocols is the \textit{required number of states}. This is a straightforward notion for simple protocols with constant number of states. For more complex protocols, transition functions are usually described by pseudo-codes, which refer to \textit{variables}. The state space of a protocol which is defined in this way is the Cartesian product of the ranges of the variables. If we consider all possible executions of a uniform protocol, then the ranges of some variables may be very large in terms of \( n \), potentially infinite. We are, however, interested in bounds on the ranges of the variables (and thus bounds on the whole state space) that hold \( w.h.p. \).

A population protocol is called \textit{uniform} if the same transition function is used for all population sizes \([7]\). Uniformity of a protocol is thus a desired property since such protocols can be applied without knowing the size of the population in advance. The original model from \([4,5]\) is uniform since the number of states of an agent is constant.

### 1.2 Related Work

It is shown in \([4,5]\) that with a constant number of states all semilinear predicates (which include, e.g., the majority predicate), can be computed in expected \( O(n^2 \log n) \) stabilization time. Recently Kosowski and Uznanski \([19]\) have shown constant-state protocols for computing the semilinear predicates and for electing a leader, \( w.h.p. \) have \( O(n \text{polylog } n) \) convergence time. Achieving fast \( O(n \text{polylog } n) \) stabilization time for the majority and leader election problems requires a growing state space, as \( O(n^{2-\varepsilon}) \) stabilization time is not possible with a constant number of states \([1,16]\).

There are a number of protocols for the majority problem and the leader election problem with \( O(n \text{polylog } n) \) stabilization time and \( O(\text{polylog } n) \) states \([1,2,3,9,12]\), but they all are non-uniform. The transition functions of these protocols depend on \( n \) as they use values \( \Omega(\log n) \) or \( \Omega(\log \log n) \). For example, each agent may be counting its interactions (which are uniform updates, not dependent on \( n \)) and progress to the next phase of the computation when its counter reaches \( \log n \) (a non-uniform update).

While the computational power of (uniform) constant-space population protocols is well understood by now, less is known about the power of uniform protocols which allow the state space to grow with the size of population. Doty et al. \([14]\) show a protocol for the exact population counting problem which has \( O(n \log n \log \log n) \) stabilization time and uses \( O(n^{60}) \) states, \( w.h.p. \) and in expectation. They also formalize the notion of uniform population protocols, modeling the agents as copies of the same Turing machine, but describe their protocols using pseudo-codes and take the size of the state space as the product of the ranges of the variables.

Doty and Eftekhari \([13]\) consider the problem of estimating the size of population within a constant factor, which they view as the problem of computing \( \log n \pm O(1) \). They show a protocol which \( w.h.p. \) has \( O(n \log^2 n) \) convergence time and uses \( O(\log^2 n \log \log n) \) states. Their protocol is not always correct and they left open the question to design a protocol which uses expected \( O(n \text{polylog } n) \) interactions and \( O(\text{polylog } n) \) states and computes \( \log n \pm O(1) \) with probability 1. An earlier work by Alistarh et al. \([4]\) includes a uniform protocol which uses \( O(n \log n) \) interactions and \( O(\log n) \) states in expectation to compute an integer which \( w.h.p. \) is between \( c_1 \log n \) and \( c_2 \log n \), for some constants \( 0 < c_1 < 1 < c_2 \). This can be viewed as approximating the population size within a polynomial factor.

### 1.3 Our Results

In this paper we present and analyze two uniform protocols, protocol \textsc{Approximate} for approximating the population size within constant factors, and protocol \textsc{CountExact} for computing the exact number of agents in the population. Unless stated otherwise we assume that all agents have to output the population size or its approximation. The theorems below summarize the performance of our protocols.

**Theorem 1.1.** Protocol \textsc{Approximate} is uniform and outputs \( \log n \) \( w.h.p. \) or \( \lceil \log n \rceil \).

\(^1\) With \textit{high probability} (\( w.h.p. \)) refers to a probability of at least \( 1 - n^{-\Omega(1)} \).
1) It converges in at most $O(n \log^2 n)$ interactions using $O(\log n \cdot \log \log n)$ states, w.h.p.

2) A variant of the protocol stabilizes in $O(n \log^2 n)$ interactions using at most $O(\log^2 n \cdot \log \log n)$ states, w.h.p.

3) Stabilization can also be achieved w.h.p. in $O(n \log^2 n)$ interactions using $O(\log n \cdot \log \log n)$ states, if not all but only $n - \log n$ agents need to output the result.

**Theorem 1.2.** The protocol COUNTEXACT is uniform and outputs the exact population size $n$. It stabilizes in $O(n \log n)$ interactions and uses $O(n)$ states, w.h.p.

Our protocols improve considerably the time and space bounds of the previous work [13] and [14]. Moreover, our approximate protocol is stable, answering the open question posed in [13], and it calculates a tighter approximation converging to $[\log n]$ or $\lceil \log n \rceil$ instead of $\log n \pm 5.7$ shown in [13].

Our algorithms are based on leader election followed by load balancing phases. A load balancing phase starts with the total load of $M$ tokens in the system and the goal is to relate $M$ to $n$ (the size of the population). If this load balancing phase completes with some agents having zero load, then w.h.p. $M \leq cn$, for a fixed constant $c > 1$. On the other hand, if all agents end up with positive load, then we must have $M \geq n$. To achieve good time and space bounds, we have to carefully control the load balancing phases and integrate them with the process of leader election. To achieve stability both of our protocols use an error detection routine and the leader initializes a process testing if the calculated answer is correct or not.

In Section 2 we discuss the auxiliary protocols which we use in our main protocols. We outline our approximate and exact protocols in Sections 3 and 4, respectively.

## 2 Auxiliary Protocols

In this section we define auxiliary protocols that we will apply in our population counting protocols: one-way epidemics, the junta process, leader election, and phase clocks.

### One-Way Epidemics.

The goal of one-way epidemics is to spread an information to all members of the population. The state space of the protocol is \{0, $x$\} for some $x > 0$. Initially at least one agent has the value $x$, which is then spread to all other agents. The transitions are formally defined as $\delta(u, v) = (\max \{u, v\}, v)$. We will refer to one-way epidemics also as broadcast. A natural extension is maximum broadcast, where agents do not only spread one possible value $x$, but instead each agent starts with its own value from the integer interval $[0, x]$. For maximum broadcast, the initiator always adopts the maximum, which is covered by the transition rule above as well. The result on one-way epidemics carries over immediately to maximum broadcast. The following result is well-known, see for example [6].

**Lemma 2.1.** Let $T_{bc}$ be the number of interactions required to complete (maximum) broadcast. W.h.p., $T_{bc} = O(n \log n)$.

### Junta Process.

The goal of the junta process [8, 18] is to mark $\Theta(n^c)$ agents – the junta. The state of each agent $v$ in this process is formed by a triplet $(\text{level}_v, \text{active}_v, \text{junta}_v) \in \mathbb{N}_0 \times \{0, 1\}^2$ initially set to $(0, 1, 1)$. The idea of the protocol is as follows. If an active agent $v$ interacts with an active agent on the same level it increases its level, otherwise it sets $\text{active}_v$ to 0. Whenever $v$ interacts with an agent on a higher level it sets $\text{junta}_v$ to 0. Inactive agents adopt the level of their communication partner if that is higher. The protocol stabilizes when all agents are inactive at the highest level. The junta is formed by all agents $v$ that reached the maximum level and have their $\text{junta}_v$ bit set to 1. The following lemma is shown in [8], here it is adapted to our setting.

---

*We define $\tilde{O}(f(n)) = O(f(n) \cdot \log^{O(1)}(f(n)))$. 

---

4
Lemma 2.2 ([8]). Let \( \text{level}^* = \max \{ \text{level}_v \} \) be the maximal level reached by the junta process. All agents become inactive within \( O(n \log n) \) interactions, \( \log \log n - 4 \leq \text{level}^* \leq \log \log n + 8 \), and the number of agents on the maximal level is \( O(\sqrt{n} \cdot \log n) \), w.h.p.

Phase Clocks. The existence of a non-empty junta of size in \([1, n^\varepsilon] \) allows the agents to synchronize themselves via so-called phase clocks [6, 18]. The phase clocks allow all agents to divide the time in phases of \( \Theta(n \log n) \) interactions each. Each agent \( v \) has a state \( \text{clock}_v \) in \( \{0, \ldots, m - 1\} \) where \( m \) is a constant. Intuitively, these clock states can be seen as the hours on a clock face. The basic idea in every interaction is that the agents always adopt the larger clock state w.r.t. the circular order modulo \( m \). Additionally, in order to keep the clock running, the members of the junta proceed one additional step when they interact with another agent on the same clock state. An agent \( v \) enters a new phase in interaction \( t \) if \( \text{clock}_v \) crosses the boundary between \( m - 1 \) and 0. In that case we say the phase clock ticks.

For easy access to the phase clocks, we equip each agent \( v \) with a variable \( \text{phase}_v \) of constant size that counts the current phase of an agent modulo some constant. Additionally, each agent \( v \) has a flag\(^3\) \( \text{firstTick}_v \). This flag is set to 1 whenever the \( \text{phase}_v \) counter is incremented, and it is set to 0 otherwise.

Let \( D_i = [D_{i}^{\text{start}}, D_{i}^{\text{end}}] \) be the interval of interactions in Phase \( i \) such that the last agent enters Phase \( i \) in interaction \( D_{i}^{\text{start}} \) and the first agent leaves Phase \( i \) in interaction \( D_{i}^{\text{end}} + 1 \).

Lemma 2.3 ([18]). For any constant \( c \geq 0 \) we can construct a phase clock using \( m = m(c) = O(1) \) states such that w.h.p. for all phases \( 1 \leq i \leq \log(n) \) we have \( cn \log n \leq D_{i}^{\text{end}} - D_{i}^{\text{start}} \leq cn \log n + \Theta(n \log n) \).

Leader Election. In [18] the authors present a stable and uniform protocol to perform leader election called \( \text{leader\_elect} \). It runs in \( O(\log n) \) phases of \( O(n \log n) \) interactions each.

The process starts with junta election in order to start two nested phase clocks. Agents perform an interaction of the outer phase clock once per phase of the inner phase clock. In the beginning, every agent runs the protocols for the phase clocks, junta election, and leader election in parallel. Whenever an agent encounters another agent on a higher (junta) level, it resets the clocks and the leader election protocol. In that way, all agents eventually run the phase clocks and the leader election process based on the junta on the highest level. The actual leader election process is quite simple: the set of leaders is halved from phase to phase, and the time is measured by the inner phase clock.

For stable leader election the authors of [18] combine their protocol with a slow protocol which is always correct in the following way: the outer phase clock runs only if at least one leader remains in the system. If at some point it counts to \( m \), w.h.p. a total of \( \Theta(n \log^2 n) \) interactions have occurred. At this time, all agents override their current leader state from the slow protocol with that of the fast protocol. When the outer phase clock ticks (i.e., it reaches \( m \)), at least one leader exists. We equip each agent \( v \) with an additional flag \( \text{leaderDone}_v \), initially set to \( \text{false} \). It is set to \( \text{true} \) as soon as the outer phase clock ticks.

Lemma 2.4 ([18]). The uniform protocol \( \text{leader\_elect} \) elects a unique leader. It stabilizes in \( O(n \log^2 n) \) interactions, using \( O(\log \log n) \) many states, w.h.p. Furthermore, after at most \( O(n \log^2 n) \) interactions all agents \( v \) have \( \text{leaderDone}_v \) set to \( \text{true} \), w.h.p., and at that time there is exactly one leader w.h.p.

Fast Leader Election. In [8] the authors describe a stable leader election protocol that admits a trade-off between the running time and the number of states. When using \( O(n) \) states, their protocol stabilizes w.h.p. in \( O(n \log n) \) interactions. In the following, we call this protocol \( \text{FastLeaderElection} \). The main idea of the protocol is to use \( O(\log n) \) random bits to reduce the number of active leaders much faster than in the original protocol from [18].

Lemma 2.5 ([8]). The uniform protocol \( \text{FastLeaderElection} \) elects a unique leader. It stabilizes in \( O(n \log n) \) interactions, using \( O(n) \) many states, w.h.p. Furthermore, after at most \( O(n \log n) \) interactions all agents have \( \text{leaderDone}_v \) set to \( \text{true} \) w.h.p.

\(^3\)Note that for all flags we use \( \text{false} \) (resp. \( \text{true} \)) and 0 (resp. 1) interchangeably.
3 Approximate Counting

In this section we show Theorem 1.1. In Section 3.1 we assume that a unique leader exists and we present a protocol which calculates $\lceil \log n \rceil$ or $\lfloor \log n \rfloor$ using a load balancing algorithm. In Section 3.2 we analyze the protocol. In Section 3.3 we show how to combine the protocol from Section 3.1 with a leader election protocol. This shows the first part of Theorem 1.1. Finally, in Section 3.4 we show how to build a stable protocol by showing the correctness of our error detection mechanism. This shows the second and the third part of Theorem 1.1.

3.1 Approximating $n$ with a Leader

In this section we assume that a unique leader is given and that all agents are synchronized via the phase clocks. The main idea of our algorithm is to inject an increasing number of tokens into the system and to use a load balancing routine to estimate the number of agents in the system. The process is finished as soon as roughly $n$ tokens are injected into the system. To save on the number of states, the agents do not store the exact number of tokens, they hold but the logarithm of that number.

Every agent $v$ stores two variables $(k_v, \text{searchDone}_v) \in \{−1, 0, 2, \ldots\} \times \{0, 1\}$, initially set to $(-1, 0)$. The variable $k_v$ stores the logarithm of the load of agent $v$, where the special value $−1$ indicates that the agent is empty. The leader $u$ orchestrates a linear search over $k_v \in \{0, 1, \ldots\}$ to find $k^*_v$ for which $\log n - 1 < k^*_v < \log n + 1$. The variable $\text{searchDone}_v$ indicates that the search is finished. The protocol runs in rounds consisting of multiple phases each. At the beginning of round $r$ the leader injects $2^r$ tokens into the system. The load is then balanced using a powers-of-two load balancing process which restricts the load of any agent to a power of two. The search stops when at the end of a round there is an agent $v$ with load larger than 1 ($k_v > 0$). At this time the leader $u$ sets $\text{searchDone}_v$ to $\text{TRUE}$ and broadcasts the value to all agents using one-way epidemics. Thus when the total number of tokens $2^r$ becomes large enough so that the load balancing leaves at least one agent with more than one token, then this number $2^r$ is taken as an approximation of the population size.

In the classical load balancing process (cf. [10]), it is assumed that $m$ indistinguishable tokens are distributed arbitrarily among $n$ agents. At interaction $t$, the load vector $L(t)$ is defined as $L(t) = (\ell_1(t), \ldots, \ell_n(t)) \in \mathbb{N}_0^n$, where $\ell_i(t)$ is the number of tokens (load) present at agent $i$ at that interaction. Assume agents $u$ and $v$ balance their loads in interaction $t \geq 1$, then $(\ell_u(t + 1), \ell_v(t + 1)) = ((\ell_u(t) + \ell_v(t))/2, \lfloor (\ell_u(t) + \ell_v(t))/2 \rfloor)$. For the powers-of-two load balancing process we define the logarithmic load vector $K = (k_1, \ldots, k_n) \in \{-1, 0, 1, \ldots\}^n$ such that for any agent $v$ we have $\ell_v(t) = 2^{k_v}$ if $k_v \geq 0$ and $\ell_v(t) = 0$ if $k_v = -1$. Assume now that agent $u$ interacts with agent $v$. A balancing action is only permitted if either $u$ or $v$ is empty (i.e., $k_u = -1$ or $k_v = -1$) and both of them are not the leader. Let $k'_u$ and $k'_v$ be the resulting logarithmic load values. Then

$$(k'_u, k'_v) = \begin{cases} (k_u - 1, k_v - 1) & \text{if } k_u > 0 \text{ and } k_v = -1 \\ (k_v - 1, k_u - 1) & \text{if } k_u = -1 \text{ and } k_v > 0 \\ (k_u, k_v) & \text{otherwise.} \end{cases}$$  \tag{1}$$

The Search Protocol is defined in Algorithm 1. The protocol runs in rounds of 5 phases. Every agent $v$ uses the variable $\text{phase}_v$ to count the number of phases modulo 5 and the flag $\text{firstTick}_v$, which is set to $\text{TRUE}$ when $v$ initiates the first interaction of a phase (see Section 2). In Phase 1 and Phase 4 the leader is active, in the remaining phases the non-leaders are active.

- Phase 0 and Phase 1 are used for the initialization. Every agent $v$ which is not the leader resets $k_v$ to $-1$ (Line 11). During the first interaction $(u, v)$ in Phase 1 the leader $u$ transfers $2^{k_u}$ tokens to $v$ (Line 3).

- In Phase 2, the non-leader agents perform powers-of-two load balancing, as described above (Line 14).

- Phase 3 consists of one-way epidemics where the agents communicate their highest load value (Line 16).

- In Phase 4 (Line 5), the leader $u$ decides if the search is finished or not. If the maximum logarithmic load is less than 1 it concludes that the injected logarithmic load was smaller than or equal to $\log n - 1$. 


Interaction \((u, v)\) in the Search Protocol:

1. if \(\text{leader}_u = \text{TRUE} \quad \text{and} \quad \text{searchDone}_u = \text{FALSE}\) then \(\triangleright \) Leader:

   2. if \(\text{phase}_u = 1 \quad \text{and} \quad \text{firstTick}_u = \text{TRUE}\) then \(\triangleright \) Phase 1: load infusion

   3. \(k_u \leftarrow k_u\)

   4. if \(\text{phase}_u = 4 \quad \text{and} \quad \text{firstTick}_u = \text{TRUE}\) then \(\triangleright \) Phase 4: decision

   5. if \(k_v \leq 0\) then

   6. \(k_v \leftarrow k_v + 1\)

   7. else

   8. \(\text{searchDone}_u \leftarrow \text{TRUE}\)

   9. if \(\text{leader}_u = \text{FALSE} \quad \text{and} \quad \text{leader}_v = \text{FALSE}\) then \(\triangleright \) Followers:

   10. if \(\text{phase}_u = 0\) then \(\triangleright \) Phase 0: initialize

   11. \(k_u \leftarrow -1\)

   12. if \(\text{phase}_u = 2\) then \(\triangleright \) Phase 2: load balancing

   13. if \(\min\{k_u, k_v\} = -1 \quad \text{and} \quad \max\{k_u, k_v\} > 0\) then

   14. \(k_u, k_v \leftarrow \max\{k_u, k_v\} - 1\)

   15. if \(\text{phase}_u = 3\) then \(\triangleright \) Phase 3: one-way epidemics

   16. \(k_u, k_v \leftarrow \max\{k_u, k_v\}\)

Algorithm 1: The Search Protocol, centerpiece of protocol Approximate

\[
s(v) = \left(\begin{array}{c}
\text{phase}_v, \quad \text{firstTick}_v, \quad \text{leader}_v, \quad k_u, \quad \text{searchDone}_v
\end{array}\right)
\]

\[
\text{Phase Clocks} \quad \quad \text{Leader Election} \quad \quad \text{Search Protocol}
\]

\[
S = \{0, \ldots, 4\} \times \{0, 1\} \times \{0, 1\} \times \{-1, 0, 1, \ldots\} \times \{0, 1\}
\]

Figure 1: State space of the Search Protocol (Algorithm 1)

The leader \(u\) therefore proceeds to the next round and injects twice as many tokens. If the observed maximum logarithmic load is larger than 1, the leader concludes the protocol by setting \(\text{searchDone}_u\) to \(\text{TRUE}\).

The number of agents is now estimated as \(2^{k_u}\). In the following, the state of an agent \(v\) in this algorithm is called \(s(v)\). The state space \(S\) is the Cartesian product of the domains of the individual variables (\(\text{phase}_v, \quad \text{firstTick}_v, \quad \text{leader}_v, \quad k_u, \quad \text{searchDone}_v\)). For an overview of the state of agent \(v\) in the Search Protocol, see Figure 1. Note that we omit internal states of constant size from auxiliary protocols.

3.2 Analysis of the Search Protocol

Based on the definition in Equation (1) we first show that the powers-of-two load balancing process on \(n\) agents balances the load such that the maximum load is at most 1 (i.e., \(k_v \leq 0\) for all agents \(v\)) as long as at most \(3/4n\) tokens are injected by the leader. Let \(k_v(t)\) be the logarithmic load of agent \(v\) at interaction \(t\).

Lemma 3.1. Assume there exists an agent \(u\) with \(k_u(0) = \kappa\) and \(k_v(0) = -1\) for all \(v \neq u\). If \(2^{\kappa} \leq 3/4 \cdot n\), then w.h.p. after \(t = 16n \log n\) interactions \(\max_v\{k_v(t)\} = 0\).

The proof of Lemma 3.1 uses the same ideas and shows a similar statement as in Lemmas 2 and 3 in [10] for the classical load balancing process.

In the following, we assume that precisely one agent \(u\) is the leader and all agents synchronize themselves via the phase clocks. Based on this assumption we can show the following lemma for the Search Protocol.
Lemma 3.2. After at most $O(\log n)$ rounds the leader $u$ sets $\text{searchDone}_u$ to true, w.h.p. At that time we have w.h.p. that $3/4 \cdot n < 2^{k_u} \leq 2\lceil \log n \rceil$.

Proof. The Search Protocol runs in multiple phases. In the following, we combine every five consecutive phases to a round such that round $r \geq 0$ consists of phases $5r$ to $5r + 4$. For a fixed round $r$ let $k_v(r)$ denote the changing value of variable $k$ of agent $v$ during round $r$. Assuming that all agents are properly synchronized by the phase clocks (see Lemma 2.3), we can w.h.p. assume that the number of interactions in any round is sufficient for spreading information with one-way epidemics (see Lemma 2.1) and for the powers-of-two load balancing (see Lemma 3.1).

Observe that the leader sets $\text{searchDone}_u$ to true in Line 8 of Phase 4 of the Search Protocol. Let $v$ be an agent with maximal load at the beginning of Phase 3. In Phase 3 the agents used one-way epidemics to disseminate the value $k_v$ to all agents. Hence, $\text{searchDone}_u$ is set to true in the first round $r$ when an agent $v$ has $k_v(r) > 0$ after the load balancing, meaning agent $v$ has a load of at least two.

To show the lemma we consider 3 cases, depending on the value $k_u(r)$ which corresponds to the load injected by the leader at the beginning of round $r \geq 0$.

Case $2^{k_u(r)} \leq 3/4 \cdot n$. In Phase 1 the leader injects $2^r$ many tokens. According to Lemma 3.1, w.h.p. at the end of Phase 2 of Round $r$ no agent will have a load larger that one, meaning for all agents $v$ we have $k_v \leq 0$. The maximum value $k_u$ which is communicated via one-way epidemics is at most zero and the leader does not set $\text{searchDone}_u$ to true in Phase 4 of Round $r$. Hence, it injects $2^{r+1}$ tokens at the beginning of the next round.

Case $3/4 \cdot n < 2^{k_u(r)} < n$. Again, in Phase 1 the leader injects $2^r$ many tokens. In this case either there exists an agent $v$ with a load larger than one ($k_v > 0$) at the end of Phase 2, or for all agents $v$ we have $k_v \leq 0$. In the latter case we are back to the previous case: the leader does not set $\text{searchDone}_u$ to true in Phase 4 of Round $r$. Hence, it injects $2^{r+1}$ tokens at the beginning of the next round. If there exists an agent $v$ with $k_v > 0$ the leader sets $\text{searchDone}_u$ to true.

Case $n \leq 2^{k_u(r)}$. In Phase 1 the leader injects $2^r \geq n$ many tokens. The agents balance at least $n$ tokens on $n - 1$ agents (since the leader does not participate in the load balancing). Hence, at the end of Phase 2 of Round $r$ there exists an agent with a load of at least two, meaning $k_v \geq 1$. The leader sets $\text{searchDone}_u$ to true in Phase 4 of Round $r$.

From the above cases it follows that after at most $\lceil \log n \rceil$ rounds $\text{searchDone}_u$ is set to true and then $3/4 \cdot n < 2^{k_u} \leq 2\lceil \log n \rceil$. \hfill \qed

3.3 Combining the Search Protocol with Leader Election

In this section we combine the Search Protocol with the LeaderElection protocol from [18]. The combined protocol works as follows. All agents run the JuntaProcess and the PhaseClocks protocols in parallel to the LeaderElection protocol and, later, the Search Protocol. In the junta process, agents cannot decide whether they have already reached the maximal junta level. Therefore, some agents already perform some interactions of the LeaderElection protocol or the Search Protocol at a lower junta level. These agents might be badly synchronized, since the phase clock ticks correctly w.h.p. only on the maximal junta level. We therefore define the following procedure that has already been used in [18]. Whenever an agent $u$ interacts with an agent $v$ in a higher level ($\text{level}_u < \text{level}_v$) all variables for PhaseClocks, LeaderElection and the Search Protocol are re-initialized. In that way, all agents eventually start the LeaderElection protocol and the Search Protocol at the maximal junta level from a clean state.

The combined protocol uses two flags $\text{leaderDone}_v$ and $\text{searchDone}_v$ for each agent $v$, initially set to false. The first flag, $\text{leaderDone}_v$, is set once an agent has concluded the leader election protocol (see Section 2). The second flag, $\text{searchDone}_v$, is set by the Search Protocol (see Section 3.1). The two flags $\text{leaderDone}_v$ and $\text{searchDone}_v$ allow every agent $v$ to distinguish between three stages of the execution of the protocol, the Leader Election Stage (1), the Search Stage (2), and the Broadcasting Stage (3).

In the Leader Election Stage all agents use the protocol from [18] to elect a leader. Recall that the flag $\text{leaderDone}_u$ is set to true once the external phase clock ticks (see Section 2). At this time there exists w.h.p.
Interaction \((u,v)\) of protocol APPROXIMATE:

1. \textbf{if} level\(_u\) > level\(_u\) \textbf{then}
2. \hspace{1em} re-initialize states of \(u\) for PHASECLOCKS, LEADERELECTION, SEARCHPROTOCOL
3. JUNTAProcess \([\{\text{level}_u, \text{junta}_u\}, \{\text{level}_v, \text{junta}_v\}]\)
4. PHASECLOCKS \([\{\text{phase}_u, \text{firstTick}_u\}, \{\text{phase}_v, \text{firstTick}_v\}]\)

5. \textbf{if not} leaderDone\(_u\) \textbf{then}
6. \hspace{1em} LEADERELECTION \([\{\text{leader}_u, \text{leaderDone}_u\}, \{\text{leader}_v, \text{leaderDone}_v\}]\)
7. \textbf{if} leaderDone\(_u\) \textbf{and not} searchDone\(_u\) \textbf{then}
8. \hspace{1em} SEARCHPROTOCOL \([\{\text{k}_u, \text{searchDone}_u\}, \{\text{k}_v, \text{searchDone}_v\}]\)
9. \textbf{if} leaderDone\(_u\) \textbf{and} searchDone\(_u\) \textbf{then}
10. \hspace{1em} (searchDone\(_v, k_e\)) ← (TRUE, k\(_u\))

\textbf{Algorithm 2:} Protocol APPROXIMATE

\begin{figure}[ht]
\centering
\begin{tabular}{cccc}
\textbf{state of} & \textbf{Junta Process} & \textbf{Phase Clocks} & \textbf{Leader Election} & \textbf{SEARCH Protocol} \\
agent \(v\): \(\text{level}_v, \text{active}_v, \text{junta}_v\) & \(\text{clock}_v, \text{phase}_v, \text{firstTick}_v\) & \(\text{leader}_v, \text{leaderDone}_v\) & \(s(v)\)
\end{tabular}
\end{figure}

\textbf{Figure 2:} State space of protocol APPROXIMATE (Algorithm 2)

exactly one leader \([18]\). In the Search Stage, we use the SEARCH Protocol defined in Section 3.1. In the Broadcasting Stage the leader \(u\) uses one-way epidemics to inform all other agents of its value \(k_u\). Each agent \(v\) uses its value \(k_v\) as its output.

We now give the proof of the first part of Theorem 1.1, where we show that protocol APPROXIMATE w.h.p. computes either \(\lceil \log n \rceil\) or \(\lceil \log n \rceil\).

\textbf{Proof of Statement 1) of Theorem 1.1.} From Lemma 2.2 it follows that after \(O(n \log n)\) interactions the junta is elected and all agents are inactive. From that point on no agent in the protocol is ever re-initialized again. Let \(u\) be the single leader that according to Lemma 2.4 w.h.p. concludes the leader election protocol. According to Lemma 3.2, the leader \(u\) sets searchDone\(_u\) to TRUE w.h.p. in \(\lceil \log n \rceil\) or \(\lceil \log n \rceil\) phases of \(O(n \log n)\) interactions each, so at that time \(k_u\) is either \(\lceil \log n \rceil\) or \(\lceil \log n \rceil\). According to Lemma 2.1, within \(O(n \log n)\) further interactions all agents know the value \(k_u\) from the leader \(u\). Together, this gives a total number of \(O(n \log^2 n)\) interactions, after which w.h.p. all agents know the value \(k_u\) from the leader and thus they all output either \(\lceil \log n \rceil\) or \(\lceil \log n \rceil\).

Regarding the required number of states, we observe that level\(_v\) from the junta process and \(k_v\) from the SEARCH Protocol are the only variables of an agent \(v\) that are not of constant size and thus may grow with the population size \(n\). For an overview of the state of agent \(v\), see Figure 2. Note that level\(_v\) is bounded w.h.p. by \(O(\log \log n)\) according to Lemma 2.2, and \(k_v\) is bounded according to Lemma 3.2 w.h.p. by \(O(\log n)\). Together this gives that w.h.p. the state space has size \(O(\log \log n \cdot \log n)\). This concludes the proof of Statement 1) of Theorem 1.1.

\subsection{3.4 The Stable Protocol}

In this section we sketch our stable protocol and proofs of Statements 2) and 3) of Theorem 1.1. The details can be found in the full version \([11]\). The protocol APPROXIMATE is not stable because there is small, but positive, probability that the system never reaches a correct stable configuration (a configuration when all agents output \(\lceil \log n \rceil\) or \(\lceil \log n \rceil\) and cannot change their output whatever future interactions may be).
stable protocol is a hybrid protocol which combines the protocol APPROXIMATE with a slow backup protocol which always finds the correct solution.

The challenge is to design mechanisms which ensure that the combined protocol stabilizes with the output from the backup protocol whenever an error occurs within the main protocol. The first error we handle in our stable protocol is concerned with leader election, where it may happen that no agent ever completes the protocol as a leader (so no one ever sets their leaderDone flag to trigger the Search Protocol), or multiple leaders do. To deal with the possibility that no agents ever set the leaderDone flag, all agents run from the very beginning the backup protocol in parallel to their main computation. The initial output is the output from this backup protocol. An agent which sets leaderDone to TRUE, stops at this time the execution of the backup protocol and proceeds with the Search Protocol, which is governed by the leader. At this time the output is switched to the output from the main protocol.

Detecting an error in the Search Protocol is more involved. We develop an ErrorDetection protocol which is executed by all agents when they complete their Search Protocol (that is, when they set their searchDone flag). The ErrorDetection protocol consists of another load balancing computation. It is a combination of the powers-of-two load balancing used in the Search Protocol, followed by the classical load balancing when the individual loads become less than a fixed constant. Combining these two load balancing computations ensures precise load balancing but requires only a constant number of additional states. The total load is set in such a way that, once the load balancing is completed, w.h.p. the load of each agent is at least 3, and no two individual loads differ by more than 2. The agents keep checking these conditions and raise an error flag whenever they find out that one of them is violated. The leader u then computes the final output based on its load, which can be shown to be either \(\lfloor \log n \rfloor\) or \(\lceil \log n \rceil\), or otherwise an error is raised at some point of the computation.

Whenever an agent detects an error, it raises an error flag, which is subsequently adopted by all other agents via one-way epidemics. When an agent sets its error flag, it ignores all its previous computations and executes a new instance of the backup protocol. For example, if multiple agents conclude the leader election stage as leaders, then this error is eventually detected when two leaders interact with each other, resulting in both raising their error flags.

The backup protocol works as follows. Every agent starts with exactly one token. Whenever two agents interact and both have the same amount of tokens, one of them hands its tokens over to the other. The agents do not store the exact number of tokens they hold but the logarithm of that number. Note that, due to the definition of the process, the load of every agent is either 0 or a power of two. After \(O(n^2 \log n)\) interactions one agent will store \(\lfloor \log n \rfloor\) tokens representing \(2^\lfloor \log n \rfloor\). For each \(0 \leq i < \lfloor \log n \rfloor\) there might be one agent storing \(2^i\) tokens (depending on the value of \(n\)). The number of states required by this process is \(O(\log n)\) if it is sufficient for all but \(\log n\) agents to know the approximation for \(n\). Otherwise, the value \(\lfloor \log n \rfloor\) has to be sent to every agent via one-way epidemics. In this case there are up to \(\log n\) many agents that need \(O(\log^2 n)\) many states (exactly those agents which store a value \(2^i\)).

The hybrid protocol constructed in this way stabilizes w.h.p. within \(O(n \log^2 n)\) many interactions using \(O(\log \log n \cdot \log^2 n)\) many states if all agents have to know the approximation of \(n\), otherwise \(O(\log \log n \cdot \log n)\) states are sufficient.

4 Counting the Exact Population Size

In this section we consider the problem of counting the exact population size and present our protocol CountExact. The main idea is as follows. The protocol first selects a junta using the modified junta process from [8] (see Section 2). It then creates phase clocks and selects a leader using the FastLeaderElection protocol (see Section 2). Again, every agent \(v\) has two flags leader, and leaderDone, which indicate whether \(v\) is a leader and \(v\) has completed the leader election protocol, respectively. Recall that the junta uses a variable level, that stores the maximum level which is reached during the junta election. In [8] it has been shown that in the modified junta process the level reaches w.h.p. a value of \(\log \log n + c\) for some constant \(c\). Hence, we can use \(2^\text{level}\) as a first approximation for \(n\). This approximation will be refined in two stages, called APPROXIMATIONSTAGE and REFINEMENTSTAGE, which are described in Section 4.1 and Section 4.2, respectively. The protocol CountExact can be found in Algorithm 3. The output of every agent in the protocol CountExact is defined by the REFINEMENTSTAGE.
Interaction \((u, v)\) of protocol CountExact:

1. if \(\text{level}_u > \text{level}_v\) then
   - re-initialize states of \(u\) for PhaseClocks, FastLeaderElection, ApproximationStage, RefinementStage

2. **JuntaProcess** \([\text{level}_u, \text{level}_v]\) \(\triangleright\) update auxiliary protocols
3. **PhaseClocks** \([\text{phase}_u, \text{firstTick}_u], (\text{phase}_v, \text{firstTick}_v)\]
4. if not leaderDone\(_u\) then
   - **FastLeaderElection** \([(\text{leader}_u, \text{leaderDone}_u), (\text{leader}_v, \text{leaderDone}_v)]\)
5. if leaderDone\(_u\) and not ApxDone\(_u\) then
   - **ApproximationStage** \([(k_u, l_u, \text{ApxDone}_u), (k_v, l_v, \text{ApxDone}_v)]\)
6. if leaderDone\(_u\) and ApxDone\(_u\) then
   - **RefinementStage** \([(k_u, l_u), (k_v, l_v)]\)

**Figure 3:** State space of protocol CountExact (Algorithm 3)

### 4.1 Fast Approximation

In this section we assume that a unique leader has been elected and that all agents are synchronized via the phase clocks. Additionally, we assume that all agents are on the same maximal junta level \(\text{level}^*\) w.r.t. the modified junta process from [8].

The goal of protocol ApproximationStage is to compute \(\log n\) up to an additive error of \(\pm 3\). (We are not using the protocol from Section 3 because we aim for the \(O(n \log n)\) bound on the number of interactions.) The protocol starts with injecting \(2^{2^{\text{level}^*}} = n^\eta\) tokens into the system for some \(0 < \eta \leq 1\). It alternates between increasing the number of injected tokens by a factor of \(n^\eta\) and load balancing until the total number of tokens \(M\) is at least \(n/2\). Unfortunately it is possible that \(M\) is very close to \(n^{1+\eta}\), resulting in a multiplicative error. Hence, the protocol outputs \(k_u = \log M - \lfloor \log l_u \rfloor\). In Lemma 4.1 we will show that \(\log n - 3 \leq k_u \leq \log n + 3\).

The interactions of the protocol are defined in Algorithm 4. The protocol runs in multiple phases of \(O(n \log n)\) interactions each (the phases are determined by the phase clock). Every agent \(v\) uses the variables \(i_v\) (initialized to \(0\)) as a phase counter and \(l_v\) for load balancing. The leader \(u\) has an additional variable \(k_u\) in it which eventually stores the approximation. In the first phase, every agent \(v\) initializes its load \(l_v\) to \(0\) (non-leaders) or to \(1\) (leader). In the first interaction of every phase, every agent \(v\) increases the phase counter \(i_v\) and multiplies its load with \(n^\eta\) (see Line 7). During the remainder of the phase, all agents us the classical load balancing process from [10] to balance their tokens. The leader \(u\) additionally checks before the multiplication whether it has a load of at least \(4\) (in which case the total load is at least \(2n\) w.h.p.). If this is the case, the leader calculates the approximation \(k_u\) as \(i_u - \eta \cdot \lfloor \log l_u \rfloor\) and raises the flag ApxDone\(_u\), indicating that the approximation stage has concluded (see Line 5). The flag is then sent to all other agents via one-way epidemics (see Line 9), and the raised flag terminates the process.

**Lemma 4.1.** Let \(u\) be the leader. After at most \(O(n \log n)\) interactions of the Approximation Stage all agents \(v\) set ApxDone\(_v\) to true w.h.p. At that time, \(k_u = \log n + 3\) w.h.p. The protocol uses w.h.p. at most \(O(n)\) states.

**Proof.** First we bound the number of phases until all agents \(v\) have ApxDone\(_v\) = true. When the leader \(u\) sets ApxDone\(_u\) to true, all other agents follow via one-way epidemics in the same phase, w.h.p. Therefore it
suffices to bound the number of phases until the leader $u$ sets $\text{ApxDone}_u$ to true.

Let $i$ be the first phase in which the leader has a load of at least 4. From load balancing [10] it follows that the total load $M$ at that time is at least $2n$. At the beginning of Phase 1 the leader $u$ has $n^i = 2^{i3\omega}/8$ tokens and all other agents are empty. In the first interaction of Phase $i > 1$ all agents $v$ multiply their load with $n^i$ (see Line 7). Hence, in Phase $i \leq i$ the total load is $M = n^{i+1}/2$, w.h.p. and in Phase $i$ we have $2n \leq n^i \leq 6 \cdot n^{1+\eta}$ (see [10]). From Lemma 2.2 we obtain that w.h.p. $\log \log n - 4 \leq \ell_u$ and therefore $\eta \geq 1/2^{12}$. Since $n^i \leq 6 \cdot n^{1+\eta}$ we get $i \leq 1/\eta + 1 + o(1)$. Hence the number of phases $i$ until $u$ sets $\text{ApxDone}_u$ to true is bounded by a constant $i = O(1)$.

We now bound the quality of the approximation $k_u$. In Phase $i$, the leader sets its variable $k_u$ to $k_u = i \cdot 2^{\ell_u} - \log M$. From load balancing [10] we get $M \leq 2^{i3\omega}/8 / n + 1.5$ and therefore

$$k_u \geq i \cdot 2^{\ell_u} - \log \left(2^{i3\omega}/8 / n + 1.5 \right)$$
$$\geq i \cdot 2^{\ell_u} - \log \left(2^{i3\omega}/8 / n \right) \cdot (1 + 1.5/2)$$
$$= i \cdot 2^{\ell_u} - \log \left(2^{i3\omega} / n \right) + \log n - \log 1.75$$
$$\geq \log n - 3,$$

where (2) holds since $2^{i3\omega}/8 \geq 2n$. Analogously,

$$k_u \leq i \cdot 2^{\ell_u} - \log \left(2^{i3\omega} / n \right) + 1$$
$$\leq i \cdot 2^{\ell_u} - \log \left(2^{i3\omega} / n \right) \cdot (1 - 1.5/2) + 1$$
$$\leq \log n + 3$$.

It remains to bound the required number of states. For each agent $v$, the variables $i_v$ and $k_v$ require w.h.p. at most $O(\log n)$ many states, and the variable $l_v$ stores up to $O(n^\theta) \leq n$ many tokens. 

### 4.2 Refining the Approximation

As before, we assume that a unique leader has been elected and that all agents are synchronized via the phase clocks. We also assume that the leader holds an approximation of $\log n \geq 3$.

In the Refinement Stage, every agent $v$ holds a variables $k_v$ for the approximation of $\log n$ and a variable $l_v$ that is used for load balancing. The protocol consists of 3 phases. The first phase is used for initialization. The value $k_u$ (the approximation calculated by the leader in the previous stage) is spread among all agents via one-way epidemics (Line 2). In the beginning of the second phase the leader injects $2^{k_u} \cdot 2^{k_v}$ many tokens into the system which are balanced, as before. In the beginning of third phase every agent multiplies its load with $2^{k_v}$, and then the load is balanced again. In Lemma 4.2 we show that total number of tokens $M$ is bounded by $2^{2 \cdot n^2} \leq M \leq 2^{14} \cdot n^2$. At the end of the phase every agent can compute $n$ as its output function $\omega(v) = [2^{8 \cdot 2^{k_v}}/l_v]$.\(^4\)

#### Lemma 4.2. After at most $O(n \log n)$ interactions of the Refinement Stage, all agents $v$ output $\omega(v) = n$ w.h.p. The protocol uses w.h.p. at most $O(n)$ states.

Proof. At the end of Phase 0, w.h.p. all agents know $k_u$ from the leader according to one-way epidemics. At the end of Phase 1, w.h.p. $C \cdot 2^{k_u}$ tokens have been balanced such that $l_u = \Theta(1)$ for any agent $v$. At the end of Phase 2, a total of at least $M \geq 4 \cdot n^2$ tokens have been balanced and every agent $v$ has load $l_v = [C \cdot 2^{2k_u}/n] \pm 1$ w.h.p.

We now need to show that every agent $v$ can output $n$ using its output function $\omega(v) = [C2^{2k_v}/l_v]$, provided that $v$ knows the value $k_v = k_u = \log n \pm 3$ from the leader and the total load $M$ is at least $M = C \cdot 2^{k_v} \geq 4n^2$ tokens. The proof is based on the proof of Lemma 3.8 from [15].

Observe that after the load balancing, w.h.p. $l_v = C \cdot 2^{2k_v} / n + r$ where $r \in [-1.5, 1.5]$ (the error term $r$ covers the remaining discrepancy and the rounding error). Let $\hat{\omega}(v)$ be the output of agent $v$ before the

\(^4\)The expression $[x]$ denotes $x$ rounded to the nearest integer, breaking ties arbitrarily.
Interaction $[(k_u, l_u, \text{ApxDone}_u), (k_v, l_v, \text{ApxDone}_v)]$ in the \textsc{ApproximationStage} protocol:

1. \textbf{if} \text{firstTick}_u = \text{TRUE} \textbf{then}
   2. \textbf{if} leader_u = \text{TRUE} \textbf{and} \ i_u = 0 \textbf{then} \quad \triangleright \text{initialize first phase}
   3. \quad \quad \ l_u \leftarrow 1
   4. \textbf{if} leader_u \text{ and } l_u \geq 4 \textbf{then} \quad \triangleright \text{found an approximation}
   5. \quad \quad \ \text{ApxDone}_u \leftarrow \text{TRUE}
   6. \quad \quad \ k_u \leftarrow i_u \cdot 2^\text{level}_u - 1 \quad \triangleright \text{start new phase: load explosion}
   7. \quad \quad \ (l_u, l_u) \leftarrow (i_u + 1, \ l_u \cdot 2^{\text{level}_u - 8})
   8. \quad \quad \ (l_u, l_v) \leftarrow ([l_u + l_v]/2), ([l_u + l_v]/2)) \quad \triangleright \text{classical load balancing}
   9. \quad \quad \ \text{ApxDone}_u \leftarrow \max \{ \text{ApxDone}_u, \text{ApxDone}_v \} \quad \triangleright \text{broadcast ApxDone}

\textbf{Algorithm 4: CountExact Approximation Stage}

Interaction $[(k_u, l_u), (k_v, l_v)]$ in the \textsc{RefinementStage} protocol:

1. \textbf{if} phase_u = 0 \textbf{then} \quad \triangleright \text{initialize agents and broadcast } k_u
   2. \quad \quad \ (k_u, k_v, l_u, l_v) \leftarrow (\max \{ k_u, k_v \}, \ \max \{ k_u, k_v \}, \ 0, 0)
   3. \textbf{if} firstTick_u = \text{TRUE} \textbf{then}
   4. \quad \quad \ \text{if} \ \text{phase}_u = 1 \text{ and leader}_u = \text{TRUE} \textbf{then} \quad \triangleright \text{the leader starts with load } 2^8 \cdot 2^{k_u}
   5. \quad \quad \quad \quad \ l_u \leftarrow 2^8 \cdot 2^{k_u}
   6. \quad \quad \ \text{if} \ \text{phase}_u = 2 \textbf{then} \quad \triangleright \text{multiply load with } 2^{k_u}
   7. \quad \quad \quad \quad \ l_u \leftarrow l_u \cdot 2^{k_u}
   8. \quad \quad \ (l_u, l_v) \leftarrow ([l_u + l_v]/2), ([l_u + l_v]/2)) \quad \triangleright \text{broadcast broadcast ApxDone}

\textbf{Algorithm 5: CountExact Refinement Stage}

Rounding is applied. We derive, analogously to [15],

$$
\hat{\omega}(v) = \frac{M}{l_v} = \frac{C \cdot 2^{2k_u}}{C \cdot 2^{2k_u}/n + r} = \frac{C \cdot 2^{2k_u} \cdot n}{C \cdot 2^{2k_u} + r n}
$$

$$
= \frac{C \cdot 2^{2k_u} \cdot n}{C \cdot 2^{2k_u} \left(1 + \frac{r n}{C \cdot 2^{2k_u}} \right)} = n \frac{C \cdot 2^{2k_u} \cdot n}{C \cdot 2^{2k_u} \left(1 + \frac{r n}{C \cdot 2^{2k_u}} \right)}
$$

which gives us, since \( r \in [-1.5, 1.5] \)

$$
\hat{\omega}(v) \geq \frac{n}{1 + \frac{1.5n}{C \cdot 2^{2k_u}}} \geq \frac{n}{1 + \frac{1.5n}{4n^2}} = n\left(1 - \frac{1}{3n + 1}\right) > n - \frac{1}{2}
$$

and

$$
\hat{\omega}(v) \leq \frac{n}{1 - \frac{1.5n}{C \cdot 2^{2k_u}}} \leq \frac{n}{1 - \frac{1.5n}{4n^2}} = n\left(1 + \frac{1}{3n + 1}\right) < n + \frac{1}{2}.
$$

Therefore, \( \omega(v) = \lfloor \hat{\omega}(v) \rfloor = n. \)
4.3 Analysis of CountExact

We now show Theorem 1.2. Due to space limitations, we only show that CountExact calculates the value of $n$ w.h.p. In the full version [11] we show that we can combine the protocol with a slow always-correct backup protocol in order to show stabilization.

Proof. From Lemma 2.2 it follows that after $O(n \log n)$ interactions the junta is elected and all agents are inactive. From that point on no agent in the protocol is ever re-initialized again. Let $u$ be the single leader that w.h.p. concludes the FastLeaderElection protocol according to Lemma 2.5 after at most $O(n \log n)$ interactions. According to Lemma 4.1, the leader $u$ computes $\log n \pm 3$ and sets $\text{ApxDone}_u$ to true after at most $O(n \log n)$ further interactions w.h.p. Finally, according to Lemma 4.2, all agents output $n$ after $O(n \log n)$ further interactions. The total number of interactions therefore is $O(n \log n)$. Observe that all protocols require at most $\tilde{O}(n)$ states (see also Figure 3). This concludes the proof.

In order to argue that CountExact stabilizes, we use the same idea as in Section 3.4, where we combined the protocol for approximate counting with a slow backup protocol which is always correct. Together with an error detection mechanism, this results in a protocol that stabilizes w.h.p. in $O(n \log n)$ interactions such that every agent outputs the exact value $n$, using $O(n)$ states. Further details can be found in the full version [11].

References


