Tight Bounds for Asynchronous Randomized Consensus

HAGIT ATTIYA AND KEREN CENSOR

Technion, Haifa, Israel

Abstract. A distributed consensus algorithm allows n processes to reach a common decision value starting from individual inputs. *Wait-free* consensus, in which a process always terminates within a finite number of its own steps, is impossible in an asynchronous shared-memory system. However, consensus becomes solvable using randomization when a process only has to terminate with probability 1. Randomized consensus algorithms are typically evaluated by their *total step complexity*, which is the expected total number of steps taken by all processes.

This article proves that the total step complexity of randomized consensus is $\Theta(n^2)$ in an asynchronous shared memory system using multi-writer multi-reader registers. This result is achieved by improving both the lower and the upper bounds for this problem.

In addition to improving upon the best previously known result by a factor of $\log^2 n$, the lower bound features a greatly streamlined proof. Both goals are achieved through restricting attention to a set of *layered* executions and using an isoperimetric inequality for analyzing their behavior.

The matching algorithm decreases the expected total step complexity by a $\log n$ factor, by leveraging the multi-writing capability of the shared registers. Its correctness proof is facilitated by viewing each execution of the algorithm as a stochastic process and applying Kolmogorov's inequality.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; G.3 [**Probability and Statistics**]—*Stochastic processes*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Distributed computing, shared-memory, lower bound, randomized algorithms, isoperimetric inequality

ACM Reference Format:

Attiya, H. and Censor, K. 2008. Tight bounds for asynchronous randomized consensus. J. ACM 55, 5, Article 20 (October 2008), 26 pages. DOI = 10.1145/1411509.1411510 http://doi.acm.org/10.1145/1411509.1411510

This research was supported by the Israel Science Foundation, Grant number 953/06.

A preliminary version of this article appeared in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, ACM, New York, 2007, pp. 155–164.

Authors' address: Taub Building, Technion Israel Institute of Technology, Haifa 32000 Israel, e-mail: {hagit,ckeren}@cs.technion.ac.il.

1. *Introduction*

Coordinating the actions of processes is crucial for virtually all distributed applications, especially in asynchronous systems. At the core of many coordination problems is the need to reach *consensus* among processes, despite the possibility of process failures. A (binary) consensus algorithm allows processes starting with input values in $\{0, 1\}$ to agree on the same output value (*agreement*); to rule out trivial solutions, this common output must be one of the inputs (*validity*). Consensus is a fundamental task in asynchronous systems, and can be employed to implement arbitrary concurrent objects [Herlihy 1991]; consensus is also a key component of the state-machine approach for replicating services [Schneider 1990; Lamport 1998].

Perhaps the most celebrated result in distributed computing shows that no deterministic algorithm can achieve consensus in an asynchronous system, if one process may fail [Fischer et al. 1985; Herlihy 1991; Loui and Abu-Amara 1987]. Due to the importance of the consensus problem, much research was invested in trying to circumvent this impossibility result. One successful approach is to allow randomized algorithms in which a nonfaulty process terminates only with probability 1. (The agreement and validity properties remain the same.) Randomized consensus algorithms are typically evaluated by their *total step complexity*, which is the expected total number of steps taken by all processes.

Many randomized consensus algorithms have been suggested, in different communication models and under various assumptions about the adversary (see Aspnes [2003]). In particular, algorithms were designed to solve randomized consensus in asynchronous shared-memory systems, against a *strong* adversary that can observe the results of local coin flips before scheduling the processes [Abrahamson 1988; Aspnes and Herlihy 1990; Aspnes 1990; Saks et al. 1991]. The total step complexity of the best previously known algorithm is $O(n^2 \log n)$ [Bracha and Rachman 1991]. A lower bound of $\Omega(\frac{n^2}{\log^2 n})$ on the expected total number of coin flips was proved by Aspnes [1998]; this implies the same lower bound on the total step complexity.

Closing the gap of $\Theta(\log^3 n)$ between the lower and the upper bounds and determining the total step complexity of randomized consensus remained an intriguing open question.

In this article, we prove that $\Theta(n^2)$ is a tight bound on the total step complexity of solving randomized consensus under the strong adversary in asynchronous shared memory systems, where processes communicate by reading and writing to multi-writer multi-reader registers.

The $\Omega(n^2)$ lower bound is obtained by considering a restricted set of schedules with a round-based structure, called *layers* [Moses and Rajsbaum 2002]. We focus on configurations at the end of each layer and classify them according to their *valence* [Fischer et al. 1985; Moses and Rajsbaum 2002], namely, the decisions that can be reached in layered extensions. Similarly to notions for deterministic algorithms, a configuration is *univalent* if there is only one possible decision value from all of the extensions of the execution from that configuration. If both decision values are possible then the configuration is *bivalent*. When a decision is reached, the configuration must be *univalent*, so the proof aims to avoid univalent configurations. As opposed to deterministic algorithms, where the valence of a configuration binds the extension to reach a certain decision value v, in a randomized algorithm the valence only implies that some execution will decide v with high probability

[Aspnes 1998]. This leaves the possibility of *null-valent* configurations, from which no decision value is reached with high probability. When the configuration is null-valent, we derive an isoperimetric inequality in order to control a one-round coin-flipping game for reaching another null-valent configuration.

We show that the lower bound is tight, by presenting a randomized algorithm for consensus, which has an $O(n^2)$ total step complexity. We give a *shared coin* algorithm with a constant agreement parameter, which leverages a single binary multi-writer register (in addition to n single-writer multi-reader registers). By viewing any schedule of the algorithm as a stochastic process, and applying Kolmogorov's inequality, we prove that for each possible decision value, all processes output that same value for the shared coin with constant probability. This can be used to obtain a randomized consensus algorithm with the same total step complexity [Aspnes and Herlihy 1990].

1.1. Related Work.

The previous $\Omega(\frac{n^2}{\log^2 n})$ lower bound [Aspnes 1998] relies on a lower bound for coin flipping games. In this proof, the adversary schedules processes step-by-step, and the results of the games are analyzed through hyperbolic functions, resulting in a long and unwieldy proof. In contrast, our approach considers only the configurations at the end of layers, allowing powerful results about product probability spaces to be applied, and streamlining the analysis of the behavior of executions.

Our general proof structure follows a proof by Bar-Joseph and Ben-Or [1998] of an $\Omega(\sqrt{n/\log n})$ lower bound on the expected number of rounds in a randomized consensus algorithm for the *synchronous message passing* model. In particular, like them, we treat null-valent configurations by considering one-round coin-flipping games and applying an isoperimetric inequality. Unlike their proof, our proof handles the more complicated shared-memory model and exploits the fact that in an asynchronous system, processes can be hidden in a one-round coin flipping game without having to fail for the rest of the execution.[1]

The layered approach was introduced by Moses and Rajsbaum [2002], who employed it to study deterministic consensus. They showed that the layered approach can unify the impossibility proof for asynchronous consensus [Fischer et al. 1985; Herlihy 1991; Loui and Abu-Amara 1987] with the lower bound on the number of rounds needed for solving synchronous consensus [Dolev and Strong 1983; Fischer and Lynch 1982]. Their work considered the message-passing model as well as the shared-memory model with *single-writer* registers. We take the layered approach one step further and extend it to randomized algorithms, deriving the lower bound on their total step complexity within the same framework as the results for deterministic algorithms. Besides incorporating randomization into the layered model, our proof also deals with the challenge of allowing processes to access *multi-writer* registers.

Abrahamson [1988] presented a randomized algorithm for solving consensus in asynchronous systems using shared memory, which had an exponential running

[1] Hiding processes in a layer can be described as a round-based model with *mobile* failures, where a process that fails in a certain round may still take steps in further rounds [Santoro and Widmayer 1989]. The equivalence between this model and the asynchronous model is discussed by Raynal and Roy [2005].

time. The first polynomial algorithm for solving randomized consensus was presented by Aspnes and Herlihy [1990]. They described an algorithm that uses a shared coin in order to reach agreement and has a total step complexity of $O(n^4)$. The amount of shared memory required by this algorithm was later bounded by Attiya et al. [1989]. Aspnes [1990] presented an algorithm for randomized consensus with $O(n^4)$ total step complexity, which also uses bounded space. These algorithms were followed by an algorithm of Saks et al. [1991] with $O(n^3)$ total step complexity and later, an algorithm of Bracha and Rachman [1991] with $O(n^2 \log n)$ total step complexity.

1.2. ORGANIZATION. In Section 2, we adapt the layered model to incorporate randomization and a strong adversary. Section 3 defines the key notions used in our lower bound proof—*potence* and *valence*—and proves that layered executions in the multi-writer shared-memory model are *potence connected*. The lower bound proof appears in Section 4, while Section 5 presents the algorithm. Section 6 contains a discussion and open problems.

2. *A Layered Model for Randomized Distributed Algorithms*

We consider a standard model of an asynchronous shared-memory system, where n processes, $p_1, \dots, p_n$, communicate by reading and writing to shared multi-writer multi-reader (atomic) registers (cf. Attiya and Welch [2004] and Lynch [1996]).

Each *step* consists of some local computation, including an arbitrary number of local coin flips (possibly biased) and one shared memory *event*, which is a read or a write to some register.

In a randomized consensus algorithm, each process p_i has an input value x_i, and should decide on an output value y_i. An algorithm for solving randomized consensus is *f-tolerant* if it satisfies the following requirements in a system where at most f processes can fail by crashing.

Agreement: For every two nonfaulty processes p_i and p_j, if y_i and y_j are assigned, then $y_i = y_j$.

Validity: For every nonfaulty process p_i, if y_i is assigned, then $y_i = x_j$ for some process p_j.

Termination: With probability 1, every nonfaulty process p_i eventually assigns a value to y_i.

A configuration C consists of the local states of all the processes, and the values of all the registers. Like many impossibility results, our proof relies on having configurations that are indistinguishable to all processes, except some set P. We denote $C \overset{\neg P}{\sim} C'$ if the state of all processes that are not in P is equal in both configurations C and C', and the values of all the registers are equal. We write $C \overset{\neg p}{\sim} C'$ when $P = \{p\}$.

For the purpose of the lower bound, we restrict our attention to a constrained set of executions, which proceed in layers. An *f-layer* is a sequence of at least $n - f$ distinct process id's. When executing a layer L, each process $p \in L$ takes a step, in the order specified by the layer.

An *f-execution* $\tau = L_1, L_2, \dots$ is a (finite or infinite) sequence of f-layers. We will consider only configurations that are reachable by finite f-executions, namely, after some finite sequence of f-layers.

Since we consider randomized algorithms, for each configuration C there is a fixed probability for every step a process will perform when next scheduled. Denote by X_i^C the probability space of the steps that process p_i will preform, if scheduled by the adversary. The probability space X_i^C depends only on the local state of p_i in configuration C, and therefore, delaying p_i does not change this probability space. Let $X^C = X_1^C \times X_2^C \times \cdots \times X_n^C$ be the product probability space. A vector $\vec{y} \in X^C$ represents a possible result of the local coin flips from a configuration C.

We assume a *strong* adversary that observes the processes' local coin flips, and chooses the next f-layer knowing what is the next step each process will take. The adversary applies a function σ to choose the next f-layer to execute for each configuration C and vector $\vec{y} \in X^C$, that is,

$$\sigma : \{(C, \vec{y}) \mid C \text{ is a configuration and } \vec{y} \in X^C\} \rightarrow \{L \mid L \text{ is an } f\text{-layer}\}.$$

When the configuration C is clear from the context, we will use the abbreviation $\sigma(\vec{y}) = L_{\vec{y}}$.

Denote by $(C, \vec{y}, L_{\vec{y}})$ the configuration that is reached by applying steps of the processes in $L_{\vec{y}}$, for a specific vector $\vec{y} \in X^C$. Then, $C \circ \sigma$ is a random variable whose values are the configurations $(C, \vec{y}, L_{\vec{y}})$, when $\vec{y}$ is drawn from the probability space X^C.

An *f-adversary* $\alpha = \sigma_1, \sigma_2, \ldots$ is a (finite or infinite) sequence of functions.

Given a configuration C and a finite prefix $\alpha_\ell = \sigma_1, \sigma_2, \ldots, \sigma_\ell$ of the adversary α, $C \circ \alpha_\ell$ is a random variable whose values are the configurations that can be reached by the algorithm. For every vector $\vec{y}_1 \in X^C$, by abuse of notation, let $\Pr[\vec{y}_1] = \Pr[\vec{y}_1 \text{ is drawn from } X^C]$ denote the probability of $\vec{y}_1$ in the probability space X^C. The probability that a configuration C' is reached is defined inductively:[2]

$$\Pr[C \circ \alpha_\ell \text{ is } C'] = \sum_{\vec{y}_1 \in X^C} \Pr[\vec{y}_1] \cdot \Pr[(C, \vec{y}_1, L_{\vec{y}_1}) \circ \alpha'_\ell \text{ is } C'],$$

where α'_ℓ is the remainder of the prefix after σ_1, that is, $\alpha'_\ell = \sigma_2, \ldots, \sigma_\ell$, and the basis of the induction for $\alpha_1 = \sigma_1$ is:

$$\Pr[C \circ \alpha_1 \text{ is } C'] = \sum_{\vec{y}_1 \in X^C} \Pr[\vec{y}_1] \cdot \ \chi_{C'}(\vec{y}_1),$$

where $\chi_{C'}(\vec{y}_1) = \chi_{C'}(C, \alpha_1, \vec{y}_1)$ characterizes whether the configuration C' is reached if $\vec{y}_1$ is drawn, that is,

$$\chi_{C'}(\vec{y}_1) = \begin{cases} 1 & (C, \vec{y}_1, L_{\vec{y}_1}) \text{ is } C' \\ 0 & \text{otherwise.} \end{cases}$$

The probability of deciding v when executing the algorithm under α from the configuration C is defined as follows: if C is a configuration in which there is a decision v, then $\Pr[\text{decision from } C \text{ under } \alpha \text{ is } v] = 1$, if C is a configuration in which there is a decision $\bar{v}$, then $\Pr[\text{decision from } C \text{ under } \alpha \text{ is } v] = 0$, otherwise,

[2] For simplicity, we assume that all the probability spaces are discrete, but a similar treatment holds for arbitrary probability spaces.

$$\Pr[\text{decision from } C \text{ under } \alpha \text{ is } v]$$

$$= \sum_{\vec{y}_1 \in X^C} \Pr[\vec{y}_1] \cdot \Pr[\text{decision from } (C, \vec{y}_1, L_{\vec{y}_1}) \text{ under } \alpha' \text{ is } v],$$

where α' is the remainder of the adversary after σ_1, that is, $\alpha' = \sigma_2, \sigma_3, \ldots$.

3. *Potence and Valence of Configurations*

In order to derive our lower bound, we are interested in the probability of reaching each of the possible decision values, from a given configuration. As the algorithm proceeds towards a decision, we expect the probability of reaching a decision to grow, where for a configuration in which a decision is reached, this probability is 1. This intuition is formalized as follows. Let $k \geq 0$ be an integer, and define

$$\epsilon_k = \frac{1}{n\sqrt{n}} - \frac{k}{(n-f)^3}.$$

Our proof makes use of adversaries that have a probability of $1 - \epsilon_k$ for reaching a certain decision value from a configuration reached after k layers. As the layer number k increases, the value of ϵ_k decreases, and the probability $1 - \epsilon_k$ required for a decision grows. The value of ϵ_k is set with foresight to achieve the stated bound.

An adversary with a high probability of deciding is defined as follows:

Definition 3.1. An f-adversary α from a configuration C that is reachable from an initial configuration by an f-execution with $k \geq 0$ layers, is v-*deciding* if $\Pr[\text{decision from } C \text{ under } \alpha \text{ is } v] > 1 - \epsilon_k$.

Next, we classify configurations according to the probabilities of reaching each of the possible decisions from them. We adapt the notion of *potence* [Moses and Rajsbaum 2002] to fit randomized algorithms.

Instead of considering all possible adversaries, we further restrict our attention to a certain subset of them, which will be specified later.

Definition 3.2. A configuration C that is reachable from an initial configuration by an f-execution with $k \geq 0$ layers, is (v, k, S)-*potent*, for $v \in \{0, 1\}$ and a set S of f-adversaries, if there is a v-deciding adversary $\alpha \in S$ from C.

Definition 3.3. A configuration is (v, k, S)-*valent* if it is (v, k, S)-potent but not $(\bar{v}, k, S)$-potent. Such a configuration is also called (k, S)-*univalent*.

A configuration is (k, S)-*bivalent* if it is both $(0, k, S)$-potent and $(1, k, S)$-potent.

A configuration is (k, S)-*null-valent* if it is neither $(0, k, S)$-potent nor $(1, k, S)$-potent.

We often say that C is v-potent (v-valent, bivalent, null-valent) *with respect to* S, when the number of layers k is clear from the context. Further, if the set S is also clear from the context, we will sometimes use the notation v-potent (v-valent, bivalent, null-valent). Figure 1 illustrates the valence of configurations as follows. A configuration C is mapped to a point in the figure, according to the maximum probabilities over all adversaries for deciding 0 and for deciding 1 from C; the valence of C is determined by the area in which the respective point lies. For

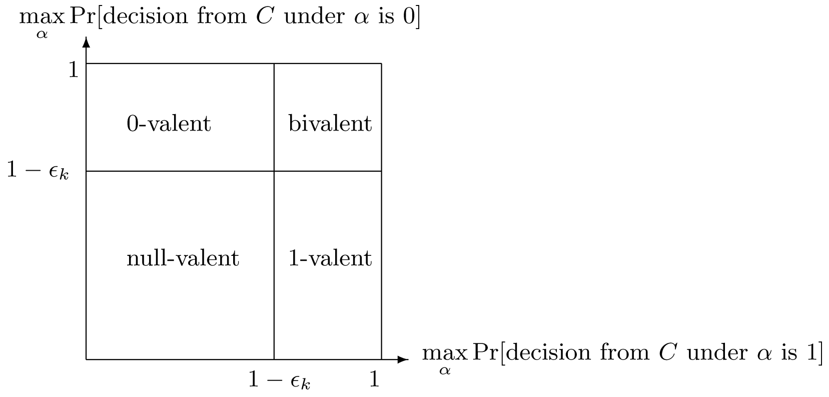
$\max_\alpha \Pr[\text{decision from } C \text{ under } \alpha \text{ is } 0]$
1
$1-\epsilon_k$
0-valent
bivalent
null-valent
1-valent
$1-\epsilon_k$
1
$\max_\alpha \Pr[\text{decision from } C \text{ under } \alpha \text{ is } 1]$

FIG. 1. Classifying configurations according to their valence.

example, if this point lies beyond $1-\epsilon_k$ on the x axis, it implies that there is an adversary σ from C with probability at least $1-\epsilon_k$ for deciding 1. Therefore, C is either bivalent or 1-valent, depending on whether it lies beyond $1-\epsilon_k$ on the y axis or not.

Note that a configuration can have a certain valence with respect to one set of adversaries S and another valence with respect to a different set S'. For example, it can be univalent with respect to S and null-valent with respect to $S' \neq S$; however, this cannot happen when $S \subseteq S'$. (Another example appears in Lemma 3.4 below.)

The set of f-adversaries we consider, denoted S_P, is induced by a subset of processes $P \subseteq \{p_1, \ldots, p_n\}$. An adversary α is in S_P, if all of the layers it may choose are P-free, where a layer is *P-free* if it does not include any process $p \in P$. Specifically, the set $S_\emptyset$ is the set of all f-adversaries.

A layer is *full with respect to S_P* if it contains the $n - |P|$ distinct process identifiers $\{p_1, \ldots, p_n\} \setminus P$; otherwise, the layer is *partial*. Scheduling a partial layer without some process $p \notin P$ does not imply that p is failed, since the system is asynchronous, and p may appear in later layers. However, considering only adversaries in S_P, for some nonempty set P, is equivalent to failing the processes in P since they do not take further steps.

Restricting a set of adversaries can only eliminate possible adversaries, and therefore cannot introduce potence that does not exist in the original set of adversaries, as formalized in the following simple lemma.

LEMMA 3.4. *If a configuration C, reached after k layers, is v-valent with respect to S_P, then it is not $\bar{v}$-potent with respect to $S_{P\cup\{p\}}$ for any process p.*

PROOF. Assume towards a contradiction, that there is a process p such that C is $\bar{v}$-potent with respect to $S_{P\cup\{p\}}$. Then, there exists a $\bar{v}$-deciding adversary α in $S_{P\cup\{p\}}$, that is,

$$\Pr[\text{decision in } C' \circ \alpha \text{ is } \bar{v}] > 1 - \epsilon_k.$$

But α is also an adversary in S_P because $S_{P\cup\{p\}} \subseteq S_P$, which implies that C is $\bar{v}$-potent also with respect to S_P, contradicting the fact that C is v-valent with respect to S_P. $\square$

Let C be a configuration reached after some number of layers k, and fix a vector $\vec{y}_1 \in X^C$. We consider every configuration that can be reached by applying a single layer to C. We define a relation between these various configurations, based on their potence, which generalizes notions for deterministic algorithms suggested by Moses and Rajsbaum [2002].

Definition 3.5. For a given vector $\vec{y}_1$, two configurations $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ have *shared potence with respect to S_P*, if they are both v-potent with respect to S_P for some $v \in \{0, 1\}$.

We define *potence connectivity* between two layers, as the transitive closure of the above relation.

Definition 3.6. For a given vector $\vec{y}_1$, two configurations $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ are *potence connected with respect to S_P*, if there is a sequence of layers $L = L_0, L_1, \ldots, L_h = L'$ such that for every i, $0 \leq i < h$, there exists a process p such that the configurations $(C, \vec{y}_1, L_i)$ and $(C, \vec{y}_1, L_{i+1})$ have shared potence with respect to $S_{P\cup\{p\}}$.

In particular, if $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ have shared potence with respect to $S_{P\cup\{p\}}$ for some process p, then they are potence connected.

Our goal is to show that given C, $\vec{y}_1$ and S_P, if the set of all configurations of the form $(C, \vec{y}_1, L)$ does not contain a null-valent configuration, then these configurations are potence connected with respect to S_P. Therefore, if there are both 0-potent and 1-potent configurations in this set, then there must also be a bivalent configuration in it. This would imply that there is a nonunivalent configuration among this set, namely, a configuration that is not v-valent, for any v.

The following claims are used to prove this connectivity, by showing that specific configurations are potence connected. They are proved under the following assumption:

Assumption 3.7. Let C be a configuration, $\vec{y}_1 \in X^C$, and S_P a set of adversaries. For every process p and every layer L, the configuration $(C, \vec{y}_1, L)$ is univalent with respect to S_P and $S_{P\cup\{p\}}$.

CLAIM 3.8. *Under Assumption* 3.7, *if $L = [p_{i_1}, p_{i_2}, \ldots, p_{i_\ell}]$ is a layer where for some j, $1 \leq j < \ell$, p_{i_j} and $p_{i_{j+1}}$ both write to the same register R, and $L' = [p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, \ldots, p_{i_\ell}]$ is the layer L after removing p_{i_j}, then $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ have shared potence with respect to $S_{P\cup\{p_{i_j}\}}$.*

PROOF. It is clear that $(C, \vec{y}_1, L) \overset{\neg P\cup\{p_{i_j}\}}{\sim} (C, \vec{y}_1, L')$, which implies that $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ have the same potence with respect to $S_{P\cup\{p_{i_j}\}}$. By Assumption 3.7, this implies that they have shared potence with respect to $S_{P\cup\{p_{i_j}\}}$, since they are not null-valent. □

CLAIM 3.9. *Under Assumption* 3.7, *if $L = [p_{i_1}, p_{i_2}, \ldots, p_{i_\ell}]$ is a layer, p is a process not in L, and $L' = [p_{i_1}, p_{i_2}, \ldots, p_{i_\ell}, p]$ is the layer L after adding p at the end, then $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ have shared potence with respect to $S_{P\cup\{p\}}$.*

PROOF. If p performs a read operation, then $(C, \vec{y}_1, L) \overset{\neg P\cup\{p\}}{\sim} (C, \vec{y}_1, L')$, which implies that these two configurations have shared potence with respect to $S_{P\cup\{p\}}$, and the claim follows.

If p performs a write operation to register R, then the states of all processes not in $P \cup \{p\}$ are the same in $(C, \vec{y}_1, L)$ and in $(C, \vec{y}_1, L')$, but the value of R may be different.

If $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ do not have shared potence with respect to $S_{P\cup\{p\}}$, then since we assume they are univalent with respect to $S_{P\cup\{p\}}$ (Assumption 3.7), we have that for some $v \in \{0, 1\}$, $(C, \vec{y}_1, L)$ is v-valent with respect to $S_{P\cup\{p\}}$ and $(C, \vec{y}_1, L')$ is $\bar{v}$-valent with respect to $S_{P\cup\{p\}}$. In particular, there is a $\bar{v}$-deciding adversary $\alpha \in S_{P\cup\{p\}}$ from $(C, \vec{y}_1, L')$. Adding p at the beginning of the first layer of α yields a new adversary which is in S_P, and is $\bar{v}$-deciding when applied to $(C, \vec{y}_1, L)$, since it is the same as applying α to $(C, \vec{y}_1, L')$. However, by Lemma 3.4, $(C, \vec{y}_1, L)$ is v-valent with respect to S_P, which is a contradiction. □

CLAIM 3.10. *Under Assumption* 3.7, *if* $L = [p_{i_1}, p_{i_2}, \ldots, p_{i_\ell}]$ *is a layer and* $L' = [p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, p_{i_j}, p_{i_{j+2}}, \ldots, p_{i_\ell}]$ *is the layer L after swapping* p_{i_j} *and* $p_{i_{j+1}}$, *then* $(C, \vec{y}_1, L)$ *and* $(C, \vec{y}_1, L')$ *are potence connected with respect to* S_P.

PROOF. If p_{i_j} and $p_{i_{j+1}}$ access different registers or if they both read, then

$$(C, \vec{y}_1, L) \overset{\neg P}{\sim} (C, \vec{y}_1, L') .$$

If p_{i_j} reads register R and $p_{i_{j+1}}$ writes to R, then

$$(C, \vec{y}_1, L) \overset{\neg P\cup\{p_{i_j}\}}{\sim} (C, \vec{y}_1, L') .$$

Both cases imply that $(C, \vec{y}_1, L)$ and $(C, \vec{y}_1, L')$ are potence connected with respect to S_P. The remaining case is when p_{i_j} and $p_{i_{j+1}}$ both write to the same register R. It may be that all the rest of the processes in the layer read from register R, and therefore distinguish between the two resulting configurations. Hence, we cannot argue in this case that the configurations have shared potence, but we can prove that they are potence connected. This is done by reverse induction on j.

Basis: If $j = \ell - 1$, let $L_0 = L$, $L_1 = [p_{i_1}, \ldots, p_{i_{\ell-2}}, p_{i_\ell}]$ be the layer L after removing $p_{i_{\ell-1}}$, and $L_2 = L'$. By Claim 3.8, $(C, \vec{y}_1, L_0)$ and $(C, \vec{y}_1, L_1)$ are potence connected with respect to S_P, and by Claim 3.9, $(C, \vec{y}_1, L_1)$ and $(C, \vec{y}_1, L_2)$ are potence connected with respect to S_P. The transitivity of potence connectivity implies that $(C, \vec{y}_1, L_0)$ and $(C, \vec{y}_1, L_2)$ are potence connected with respect to S_P.

Induction step: Let

$$L_0 = L = [p_{i_1}, p_{i_2}, \ldots, p_{i_\ell}]$$

and let

$$L_1 = [p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, p_{i_{j+2}}, \ldots, p_{i_\ell}]$$

be the layer L_0 after removing p_{i_j}. By Claim 3.8, $(C, \vec{y}_1, L_0)$ and $(C, \vec{y}_1, L_1)$ are potence connected with respect to S_P. Let

$$L_2 = [p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, p_{i_{j+2}}, \ldots, p_{i_\ell}, p_{i_j}]$$

be the layer L_1 after adding p_{i_j} at the end. By Claim 3.9, $(C, \vec{y}_1, L_1)$ and $(C, \vec{y}_1, L_2)$ are potence connected with respect to S_P.

For every m, $3 \le m \le \ell - j + 1$, let

$$L_m = [p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, p_{i_{j+2}}, \ldots, p_{i_j}, p_{i_{\ell-m+3}}, \ldots, p_{i_\ell}]$$

$L_0 =$	p_1:r(R_1)	$\ldots$	p_i:w(R_2)	p_{i+1}:r(R_2)	p_{i+2}:w(R_2)	$\ldots$	p_{n-1}:r(R_3)	p_n:r(R_4)
$L' =$	p_1:r(R_1)	$\ldots$	p_i:w(R_2)	p_{i+1}:r(R_2)	p_{i+2}:w(R_2)	$\ldots$	p_{n-1}:r(R_3)	
$L'' =$	p_1:r(R_1)	$\ldots$	p_{i+1}:r(R_2)	p_i:w(R_2)	p_{i+2}:w(R_2)	$\ldots$	p_{n-1}:r(R_3)	
$L_1 =$	p_1:r(R_1)	$\ldots$	p_{i+1}:r(R_2)		p_{i+2}:w(R_2)	$\ldots$	p_{n-1}:r(R_3)	

FIG. 2. Example of potence connected configurations: the first and second configuration are connected by Claim 3.9, the second and third are connected by Claim 3.10, while the third and fourth are connected by Claim 3.8.

be the previous layer L_{m-1} after swapping p_{i_j} with the process before it, until it reaches $p_{i_{j+1}}$. Specifically,

$$L_{\ell-j+1} = L' = [p_{i_1}, \ldots, p_{i_{j-1}}, p_{i_{j+1}}, p_{i_j}, p_{i_{j+2}}, \ldots, p_{i_\ell}].$$

By the induction hypothesis, $(C, \vec{y}_1, L_m)$ and $(C, \vec{y}_1, L_{m+1})$ are potence connected with respect to S_P, for every m, $2 \le m < \ell - j + 1$. The transitivity of potence connectivity implies that $(C, \vec{y}_1, L_0)$ and $(C, \vec{y}_1, L_{\ell-j+1})$ are potence connected with respect to S_P. $\square$

Figure 2 shows an example of using the claims, for the full layer L_0 and the partial layer L_1 obtained by removing the steps of p_i and p_n from L_0. We show two layers L' and L'' such that only one process, p_n, distinguishes between the configurations lead to by L_0 and L', only one process, p_{i+1}, distinguishes between the configurations lead to by L' and L'', and only one process, p_i, distinguishes between the configurations lead to by L'' and L_1. Thus, the configurations lead to by L_0 and L_1 must be potence connected.

The following lemma shows that given a configuration C and a vector $\vec{y} \in X^C$, if there is a layer that extends C into a v-valent configuration and a layer that extends C into a $\bar{v}$-valent configuration, then we can find a layer that extends C into a non-univalent configuration, possibly by failing one additional process.

LEMMA 3.11. *Let C be a configuration and let $\vec{y}_1$ be a vector in X^C. If there are layers L_v and $L_{\bar{v}}$, such that $(C, \vec{y}_1, L_v)$ is $(v, k+1, S_P)$-valent and $(C, \vec{y}_1, L_{\bar{v}})$ is $(\bar{v}, k+1, S_P)$-valent, then there is a layer L such that $(C, \vec{y}_1, L)$ is either not $(k+1, S_P)$-univalent or not $(k+1, S_{P\cup\{p\}})$-univalent, for some process p.*

PROOF. Assume towards a contradiction that for every layer L and every process p, the configuration $(C, \vec{y}_1, L)$ is univalent with respect to both S_P and $S_{P\cup\{p\}}$ (this implies that Assumption 3.7 holds). Let L^F be the full layer with respect to S_P consisting of all processes not in P, according to the order of their id's. Then, L^F is univalent with respect to S_P, say it is $(\bar{v}, k+1, S_P)$-valent. (Otherwise, we follow the same proof with $L_{\bar{v}}$.)

Denote $L_v = [p_{i_1}, \ldots, p_{i_\ell}]$ and consider the layer $L' = [p_{i_1}, \ldots, p_{i_\ell}, \ldots]$ that is full with respect to S_P, and has L_v as a prefix. (L_v may be full with respect to S_P, in which case L' is equal to L_v.)

We start with the layer L^F and repeatedly swap processes until we reach the layer L', in a chain of configurations which, by Claim 3.10, are potence connected with respect to S_P. From L', we repeatedly remove the last process until reaching the layer L_v, in a chain of configurations which, by Claim 3.9, are potence connected with respect to S_P. This implies that $(C, \vec{y}_1, L^F)$ and $(C, \vec{y}_1, L_v)$ are potence connected with respect to S_P.

Since $(C, \vec{y}_1, L_v)$ is $(v, k+1, S_P)$-valent, and $(C, \vec{y}_1, L^F)$ is $(\bar{v}, k+1, S_P)$-valent, it follows that there are layers L_1 and L_2 such that $(C, \vec{y}_1, L_1)$ is $(v, k+1, S_P)$-valent, $(C, \vec{y}_1, L_2)$ is $(\bar{v}, k+1, S_P)$-valent, and $(C, \vec{y}_1, L_1)$ and $(C, \vec{y}_1, L_2)$ have shared potence with respect to $S_{P\cup\{p\}}$, for some process p. By Lemma 3.4 and our assumption that all layers lead to univalent configurations, $(C, \vec{y}_1, L_1)$ is $(v, k+1, S_{P\cup\{p\}})$-valent, and $(C, \vec{y}_1, L_2)$ is $(\bar{v}, k+1, S_{P\cup\{p\}})$-valent, and hence, they cannot have shared potence with respect to $S_{P\cup\{p\}}$. This yields a contradiction and proves the lemma. □

4. *The Lower Bound*

Before presenting the lower bound proof, let us recall lower bound proofs and impossibility results for deterministic algorithms. In these proofs, the configurations are classified into *univalent* and *bivalent* configurations. Since a deciding configuration has to be univalent, these proofs aim to avoid univalent configurations by showing that there is an initial bivalent configuration, and by moving from one bivalent configuration to another bivalent configuration.

Our proof generalizes the above technique to randomized algorithms as follows. Recall that in addition to bivalent and univalent configurations, we have null-valent configurations, since valence is now a probabilistic notion. We first show that some initial configuration is not univalent (Lemma 4.1); namely, it is either bivalent or null-valent.

Ideally, we would like to complete the proof by showing that a nonunivalent configuration can be extended by a single layer to a nonunivalent configuration, while (permanently) failing at most one more process. Doing so would allow us to construct a layered execution with f layers, each containing at least $n - f$ process steps, which implies the desired lower bound.

In Lemma 4.7 (Section 4.4), we show that this can be done with high probability in the case of null-valent configurations, that is, we can extend a null-valent configuration by one layer and reach another null-valent configuration.

A bivalent configuration, can be extended with both a v-deciding adversary and a $\bar{v}$-deciding adversary, which we would like to use in Lemma 3.11 to obtain a nonunivalent configuration. However, some complications arise here, which are taken care of in Lemmas 4.3 and 4.4 (Section 4.2).

We extend the execution in this manner, with high probability, for f layers. Since $n - f$ processes take a step in each layer, we obtain the bound of an expected $\Omega(f(n - f))$ steps (Theorem 4.8).

4.1. Initial Configurations.

We start by applying Lemma 3.4 to show that some initial configuration is not univalent.

LEMMA 4.1. *There exists an initial configuration C that is not univalent with respect to either $S_\emptyset$ or $S_{\{p\}}$, for some process p.*

PROOF. Assume that all initial configurations are univalent with respect to $S_\emptyset$. Consider the initial configurations $C_0, C_1, \ldots, C_n$ such that in C_i, $0 \le i \le n$, the input of process p_j is 1 if $j \le i$ and 0, otherwise. By the validity condition, C_0 is $(0, 0, \emptyset)$-valent and C_n is $(1, 0, \emptyset)$-valent. Therefore, there is an i, $1 \le i \le n$, such that C_{i-1} is $(0, 0, \emptyset)$-valent and C_i is $(1, 0, \emptyset)$-valent. Clearly, $C_{i-1} \overset{\neg p_i}{\sim} C_i$, and hence, C_{i-1} and C_i have the same valence with respect to $S_{\{p_i\}}$. By Lemma 3.4, C_{i-1} is not 1-potent with respect to $S_{\{p_i\}}$, and C_i is not 0-potent with respect to $S_{\{p_i\}}$. Hence, they are null-valent with respect to $S_{\{p_i\}}$. $\square$

4.2. BIVALENT AND SWITCHING CONFIGURATIONS. As mentioned before, from a bivalent configuration we have both a v-deciding adversary and a $\bar{v}$-deciding adversary. However, we cannot use them directly in Lemma 3.11 to obtain a non-univalent configuration, since the first layer of a v-deciding adversary may still lead to a $\bar{v}$-valent configuration, because these definitions are probabilistic. If ϵ_k was an increasing function of k, the above situation would have small enough probability in order to simply neglect it. However, this cannot be done, since ϵ_k is defined as a decreasing function of k, in order to handle the null-valent configurations.

Instead, we prove (Lemma 4.3) that by failing at most one more process, a bivalent configuration can be extended by a single layer to a non-univalent configuration, or to a configuration which is $\bar{v}$-valent although reached while following a v-deciding adversary. Such a configuration, as will be formalized below, is called $\bar{v}$*-switching*.

We also prove that there is a small probability of deciding in a switching configuration and thus, from a switching configuration, the execution can be extended (with high probability) to a non-univalent configuration, by at least one layer (Lemma 4.4).

We formally define switching configurations as follows:

Definition 4.2. Let C be a (v, k, S_P)-potent configuration, let $\alpha = \sigma_1, \sigma_2, \ldots$ be a v-deciding adversary from C in S_P, and let $\vec{y}_1$ be a vector in X^C such that the configuration $(C, \vec{y}_1, \sigma_1(\vec{y}_1))$ is $(\bar{v}, k+1, S_P)$-valent. Then $(C, \vec{y}_1, \sigma_1(\vec{y}_1))$ is a $\bar{v}$*-switching configuration with respect to S_P from C by $\vec{y}_1$ and α.*

Lemma 4.3 implies that a bivalent configuration can be extended with one layer to a configuration that is either switching or nonunivalent.

LEMMA 4.3. *If a configuration C is (k, S_P)-bivalent, then there is an adversary σ such that for every vector $\vec{y}_1 \in X^C$, $(C, \vec{y}_1, \sigma(\vec{y}_1))$ is either $\bar{v}$-switching for some $v \in \{0, 1\}$, or not $(k+1, S_P)$-univalent or not $(k+1, S_{P\cup\{p\}})$-univalent, for some process p.*

PROOF. Assume that for every layer L and every process p, the configuration $(C, \vec{y}_1, L)$ is univalent with respect to S_P and to $S_{P\cup\{p\}}$.

Consider the extension of C with L^F, the full layer with respect to S_P. Fix a vector $\vec{y}_1 \in X^C$ and assume that $C'' = (C, \vec{y}_1, L^F)$ is $(\bar{v}, k+1, S_P)$-valent. Since C is bivalent, there is a v-deciding adversary $\alpha = \sigma_1, \sigma_2, \ldots$ in S_P. Consider the configuration $C' = (C, \vec{y}_1, \sigma_1(\vec{y}_1))$. By the assumption, C' is univalent. If it is $(\bar{v}, k+1, S_P)$-valent then it is $\bar{v}$-switching with respect to S_P from C by $\vec{y}_1$ and α. Otherwise, it is $(v, k+1, S_P)$-valent. Since C'' is $(\bar{v}, k+1, S_P)$-valent, by Lemma 3.11, there exists a layer L and a process p such that $(C, \vec{y}_1, L)$ is either not $(k+1, S_P)$-univalent or not $(k+1, S_{P\cup\{p\}})$-univalent. $\square$

The main issue when reaching a $\bar{v}$-switching configuration is that we cannot rule out the possibility that all the other layers lead to $\bar{v}$-valent configurations as well. However, although a $\bar{v}$-switching configuration is $\bar{v}$-valent, the adversary that leads to it is v-deciding. This allows us to look several layers ahead, for the desired situation of one layer leading to a v-valent configuration, and another layer leading to a $\bar{v}$-valent configuration. From such a setting, Lemma 3.11 can be used to reach a non-univalent configuration again. The following lemma presents the details of this argument.

LEMMA 4.4. *Let C' be a $\bar{v}$-switching configuration with respect to S_P from C by $\vec{y}_1$ and α. Then, with probability at least $1 - \frac{1}{n\sqrt{n}}$, C' can be extended with at least one layer to a configuration that is either v-switching, or not univalent with respect to S_P, or not univalent with respect to $S_{P\cup\{p\}}$, for some process p.*

PROOF. Let $\alpha = \sigma_1\sigma_2\cdots$; note that $C' = (C, \vec{y}_1, \sigma_1(\vec{y}_1))$. Denote $C_0 = C$, $C_1 = C'$ and for every $k \geq 2$, fix a vector $\vec{y}_k \in X^{C_{k-1}}$ and let $C_k = (C_{k-1}, \vec{y}_k, \sigma_k(\vec{y}_k))$.

Assume that for every $k \geq 1$, C_k is univalent, and let C_ℓ be the first configuration that is v-valent with respect to S_P. If such a configuration does not exist then the execution either reaches a configuration that decides $\bar{v}$, or does not reach any decision. Since α is v-deciding from C, the probability that C_ℓ does not exist is at most $\epsilon_k \leq \frac{1}{n\sqrt{n}}$.

Since C_ℓ is the first configuration which is v-valent, $C_{\ell-1}$ is $\bar{v}$-valent. Therefore, there is a $\bar{v}$-deciding adversary $\beta = \rho_1, \rho_2, \ldots$ from $C_{\ell-1}$ in S_P. Consider the configuration $C'' = (C_{\ell-1}, \vec{y}_\ell, \rho_1(\vec{y}_\ell))$. If C'' is not $(k+\ell, S_P)$-univalent, then we are done. If C'' is $\bar{v}$-valent, then since C_ℓ is v-valent, by Lemma 3.11, there exists a layer L such that $(C_{\ell-1}, \vec{y}_\ell, L)$ is either not $(k+\ell, S_P)$-univalent or not $(k+\ell, S_{P\cup\{p\}})$-univalent, for some process p, in which case we are also done. Otherwise, C'' is v-valent, which implies that it is v-switching with respect to S_P from $C_{\ell-1}$ by $\vec{y}_\ell$ and β. $\square$

4.3. ONE-ROUND COIN-FLIPPING GAMES.

The remaining part of the lower bound proof deals with null-valent configurations, and it relies on results about one-round coin-flipping games. As defined by Bar-Joseph and Ben-Or [1998], a *U-valued one-round coin flipping game of m players* is a function

$$g : \{X_1 \cup \bot\} \times \{X_2 \cup \bot\} \times \cdots \times \{X_m \cup \bot\} \longrightarrow \{1, 2, \ldots, U\},$$

where X_i, $1 \leq i \leq m$, is the ith probability space. A *t-hiding adversary* may hide at most t of the random choices in $X_1, \ldots, X_m$, by replacing them with a $\bot$.

Before presenting the formal definitions for the adversary and the game, we describe one main difference in the way that null-valent and bivalent configurations are handled. We have shown in Section 4.2 that in order to reach a bivalent configuration from a bivalent configuration, it suffices to fail one process per layer. This process is failed permanently, and may not take steps in any later layer. The case of a null-valent configuration is different. We may need to hide many more processes in a layer in order to reach another null-valent configuration. Fortunately, these processes are only *hidden* in this layer and may take steps in subsequent layers, which implies that they are not failed according to the definition of an asynchronous system. Therefore, we do not need to account for them towards the f processes that we are allowed to fail, and hence their number can be non-constant.

Formally, let $X = X_1 \times \cdots \times X_m$ be the product probability space implied by the game. For every vector $\vec{y} \in X$, and $I \subseteq \{1, \ldots, m\}$, the vector reached when the adversary hides the coordinates of I is defined as follows:

$$\vec{y}_I(i) = \begin{cases} \vec{y}(i), & i \notin I \\ \bot, & i \in I. \end{cases}$$

For every possible outcome of the game $u \in \{1, \ldots, U\}$, define the set of all vectors in X for which no t-hiding adversary can force the outcome of the game to be u, to be

$$W^u = \{\vec{y} \in X | g(\vec{y}_I) \neq u \text{ for every } I \subseteq \{1, \ldots, m\} \text{ such that } |I| \leq t\}.$$

We prove that when $|U| = 3$, there is an outcome for which there is high probability for hiding values in a way that forces this outcome; that is, for some $u \in \{1, 2, 3\}$, $\Pr[\vec{y} \in W^u]$ is very small. The proof relies on an isoperimetric inequality, following a result of Schechtman [1981]. The idea is to show that if all three sets $W^1, W^2, W^3 \subseteq X$ have large enough probability, then there is a non-zero probability for an element $\vec{z} \in X$ that is close to all three sets according to the Hamming distance. This will imply the existence of a t-hiding adversary for $\vec{z}$, for which the value of the game cannot be any of 1, 2 or 3, and hence there is a point for which the game is undefined.

Formally, the space (X, d) is a finite metric space where for every pair of vectors $\vec{x}$ and $\vec{y}$ in X, $d(\vec{x}, \vec{y})$ is the Hamming distance between $\vec{x}$ and $\vec{y}$ (the number of coordinates in which $\vec{x}$ and $\vec{y}$ differ). For $A \subseteq X$, $B(A, t)$ is the *ball of radius t around the set A*, that is,

$$B(A, t) = \{\vec{y} \in X | \text{ there is } \vec{z} \in A \text{ such that } d(\vec{y}, \vec{z}) \leq t\}.$$

The next lemma is the isoperimetric inequality we rely on. We show that given the probability of a set A, there is a lower bound on the probability of the ball of a certain radius around A.

LEMMA 4.5. *Let $X = X_1 \times \cdots \times X_m$ be a product probability space and $A \subseteq X$ such that $Pr[\vec{x} \in A] = c$. Let $\lambda_0 = \sqrt{2m \log \frac{2}{c}}$, then for $\ell \geq \lambda_0$,*

$$Pr[\vec{x} \in B(A, \ell)] \geq 1 - 2e^{-\frac{(\ell-\lambda_0)^2}{2m}}.$$

PROOF. Consider an element $\vec{x}$ as a random function $\vec{x} : D = \{1, \ldots, m\} \to X_1 \cup \cdots \cup X_m$ such that $\vec{x}(i) \in X_i$. Define a sequence of partial domains $\emptyset = D_0 \subset D_1 \subset \cdots \subset D_m = D$ such that $D_i = \{1, \ldots, i\}$. Let $f : X \to \mathbb{R}$ be a function that measures the distance of elements from the given subset $A \subseteq X$, i.e., $f(\vec{x}) = d(A, \vec{x})$.

Choose a random element of $\vec{w} \in X$ according to the given distribution. Define the sequence $Y_0, \ldots, Y_m$ by $Y_i = \mathrm{E}[f(\vec{x}) \mid \vec{x}|_{D_i} = \vec{w}]$. Specifically, $Y_0 = \mathrm{E}[f(\vec{x})]$ with probability 1, and $Y_m = f(\vec{w})$ with the probability of choosing $\vec{w}$. It is well known that $Y_0, \ldots, Y_m$ is a Doob martingale (see Alon and Spencer [2000, Chap. 7] and Motwani and Raghavan [1995, Chap. 4]).

Notice that X_1, ..., X_m are independent and therefore for every $i \in D$, the random variable $\vec{x}(i)$ is independent of other values of $\vec{x}$.

The function f satisfies the Lipschitz condition, that is, for every 2 vectors $\vec{x}, \vec{y}$ that differ only in $D_i \backslash D_{i-1}$ for some i, we have $|f(\vec{x}) - f(\vec{y})| \leq 1$, by the triangle inequality of the Hamming metric d. This implies that the martingale $Y_0, \ldots, Y_m$ satisfies the martingale Lipschitz condition, that is, $|Y_i - Y_{i-1}| \leq 1$ for every i, $0 < i \leq m$.

By Azuma's inequality, we have that for every real number $\lambda > 0$

$$\Pr[|f(\vec{x}) - \mathrm{E}[f(\vec{x})]| > \lambda] < 2e^{-\frac{\lambda^2}{2m}}. \tag{1}$$

We now claim that $\mathrm{E}[f(\vec{x})] \leq \lambda_0$. Assume the contrary, that $\mathrm{E}[f(\vec{x})] > \lambda_0$. Since $\lambda_0 = \sqrt{2m \log \frac{2}{c}}$, we have that $2e^{-\frac{\lambda_0^2}{2m}} = c$. For every $\vec{x} \in A$, we have $f(\vec{x}) = 0$, therefore

$$\Pr[|f(\vec{x}) - \mathrm{E}[f(\vec{x})]| > \lambda_0] \geq \Pr[f(\vec{x}) = 0] = c,$$

contradicting (1).

Hence, for every $\ell \geq \lambda_0$, we have

$$\begin{aligned}\Pr[\vec{x} \notin B(A, \ell)] = \Pr[f(\vec{x}) > \ell] &\leq \Pr[|f(\vec{x}) - \mathrm{E}[f(\vec{x})]| > \ell - \lambda_0] \\ &< 2e^{-\frac{(\ell-\lambda_0)^2}{2m}},\end{aligned}$$

which completes the proof. □

We now use Lemma 4.5 to show that one of the sets W^u has a small probability, and deduce that for $u = 1, 2$ or 3 the outcome of the game can be forced to be u with high probability.

LEMMA 4.6. *For every* 3*-valued one-round coin-flipping game of m players, there is a t-hiding adversary, $t = 6\sqrt{2m \log(m^3)}$, that can force the outcome of the game to be some $u \in \{1, 2, 3\}$ with probability greater than $1 - \frac{1}{m^3}$.*

PROOF. Recall that for every $u \in \{1, 2, 3\}$, the set W^u is the set of all vectors in X for which no t-hiding adversary can force the outcome of the game to be u. We wish to prove that $\Pr[\vec{y} \in W^u] < \frac{1}{m^3}$, for some $u \in \{1, 2, 3\}$.

Denote $B^u = B(W^u, \frac{t}{3})$. Assume that $\Pr[\vec{y} \in W^u] \geq \frac{1}{m^3}$ for all $u \in \{1, 2, 3\}$. Clearly, $\bigcap_{u \in \{1,2,3\}} W^u = \emptyset$, since the value of the game is undefined for points in the intersection. Moreover, we claim that $\bigcap_{u \in \{1,2,3\}} B^u$ is empty.

Assume there is a point $\vec{z} \in \bigcap_{u \in \{1,2,3\}} B^u$ (see Figure 3). For every $u \in \{1, 2, 3\}$, since $\vec{z} \in B^u$, there is a point $\vec{x}^u \in W^u$ and a set of indices $I_u \subseteq \{1, \ldots, m\}$, such that $|I_u| \leq \frac{t}{3}$ and $\vec{z}_{I_u} = \vec{x}^u_{I_u}$. Let $I = \bigcup_{u \in \{1,2,3\}} I_u$. Since $\vec{x}^u \in W^u$, we have that $g(\vec{x}^u_I) \neq u$, and hence $g(\vec{z}_I) \neq u$. This implies that $g(\vec{z}_I)$ is undefined, and therefore $\bigcap_{u \in \{1,2,3\}} B^u = \emptyset$.

We now apply Lemma 4.5 for every $u = 1, 2, 3$, with $A = W^u$. Notice that the results of the local flips of each player are independent random variables. We have that $\Pr[\vec{y} \in W^u] \geq c$ where $c = \frac{1}{m^3}$, and therefore $\lambda_0 = \sqrt{2m \log(2m^3)}$. Hence, for $\ell = \frac{t}{3} = 2\sqrt{2m \log(2m^3)} = 2\lambda_0$, we have

$$\begin{aligned}\Pr[\vec{y} \in B^u] \geq 1 - 2e^{-\frac{(t-\lambda_0)^2}{2m}} &= 1 - 2e^{-\frac{2m \log(2m^3)}{2m}} \\ &= 1 - 2e^{-\log(2m^3)} \\ &= 1 - \frac{1}{m^3}.\end{aligned}$$

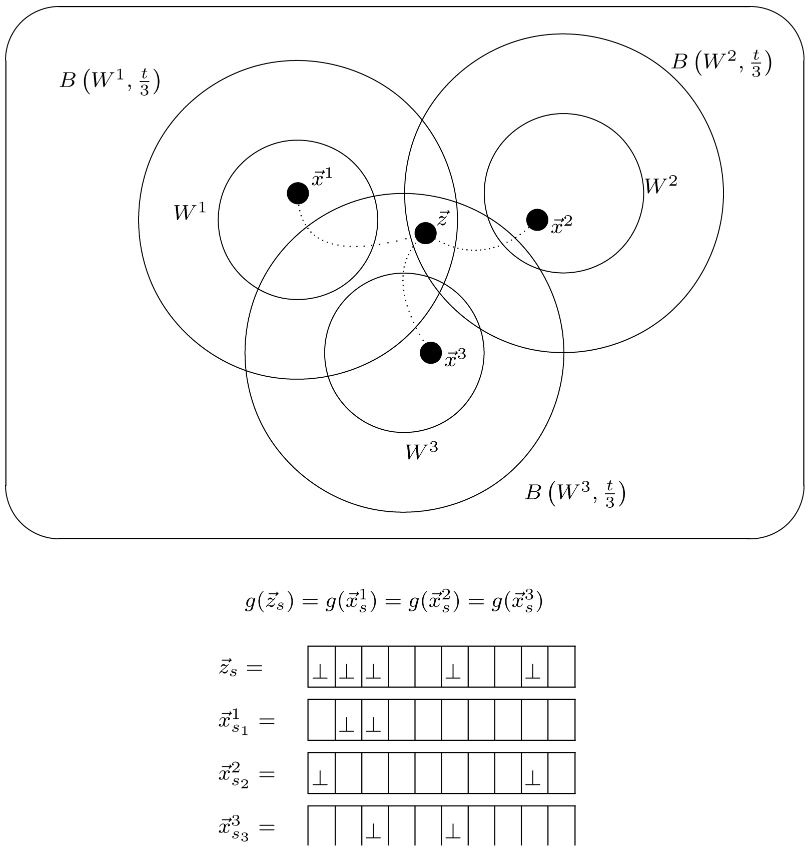

FIG. 3. The probability space $X = X_1 \times X_2 \times \cdots \times X_m$. The distance between the point $\vec{z}$ to each of the points $\vec{x}^1, \vec{x}^2, \vec{x}^3$ is at most $\frac{t}{3}$.

Since $\bigcap_{u\in\{1,2,3\}} B^u = \emptyset$, we have that

$$\Pr[\vec{y} \in B^1 \cap B^2] + \Pr[\vec{y} \in B^1 \cap B^3] + \Pr[\vec{y} \in B^2 \cap B^3] \le \frac{1}{2} \cdot \sum_{u\in\{1,2,3\}} \Pr[\vec{y} \in B^u],$$

which implies that

$$\begin{aligned}\Pr[\vec{y} \in \cup_{u\in\{1,2,3\}} B^u] &= \sum_{u\in\{1,2,3\}} \Pr[\vec{y} \in B^u] - \sum_{u\neq u'\in\{1,2,3\}} \Pr[\vec{y} \in B^u \cap B^{u'}] + \Pr\left[\vec{y} \in \bigcap_{u\in\{1,2,3\}} B^u\right] \\ &\ge \frac{1}{2} \cdot \sum_{u\in\{1,2,3\}} \Pr[\vec{y} \in B^u] \\ &\ge \frac{3}{2} \cdot \left(1 - \frac{1}{m^3}\right) > 1.\end{aligned}$$

This contradiction implies that for some $u \in \{1, 2, 3\}$, we have $\Pr[\vec{y} \in W^u] < \frac{1}{m^3}$. $\square$

4.4. NULL-VALENT CONFIGURATIONS. In the final stage of the lower bound construction, we use one-round coin-flipping games to show that, with high probability, a null-valent configuration C can be extended with one f-layer to a null-valent configuration. In order to achieve the above, we may need to hide up to $6\sqrt{2n\log(2n^3)}$ processes, other than the processes in P, in the layer. Therefore, we assume that $f \geq 12\sqrt{2n\log(2n^3)}$, and will always make sure that $|P| \leq \frac{f}{2}$. This will allow us to hide $\frac{f}{2} \geq 6\sqrt{2n\log(2n^3)}$ additional processes (not in P), in executions in S_P.

LEMMA 4.7. *If a configuration C reachable by an f-execution is (k, S_P)-null-valent, then with probability at least $1 - \frac{1}{(n-|P|)^3}$, there is an f-adversary σ_1 such that $C \circ \sigma_1$ is $(k+1, S_P)$-null-valent.*

PROOF. Let C be a (k, S_P)-null-valent configuration. We consider every vector $\vec{y}_1 \in X^C$ and every layer L in S_P, and classify the resulting configurations $(C, \vec{y}_1, L)$ into three disjoint categories:

(1) The configuration $(C, \vec{y}_1, L)$ is $(0, k+1, S_P)$-potent.
(2) The configuration $(C, \vec{y}_1, L)$ is $(1, k+1, S_P)$-valent.
(3) The configuration $(C, \vec{y}_1, L)$ is $(k+1, S_P)$-null-valent.

Notice that the first category contains both $(0, k+1, S_P)$-valent and $(k+1, S_P)$-bivalent configurations.

This can be viewed as a 3-valued one-round coin-flipping game of m players, as follows. The m players are the processes not in S_P, that is, $m = n - |P|$. The probability spaces $X_1, \ldots, X_m$ represent the random choices of the processes, which are given by $\vec{y}_1$. Every vector of choices induces a resulting configuration $(C, \vec{y}_1, L^F)$, where L^F is the full layer with respect to S_P, in which the processes take a step in the order of their identifiers. Hiding an element X_i by the adversary is done by choosing a partial layer L in S_P that does not contain any step by the corresponding processes, but only a step of each process that is not hidden. Finally, the value of the game is the category of the configuration $(C, \vec{y}_1, L)$.

Since $m = n - |P|$, we have that $n - f \leq m \leq n$. By the coin flipping game in Lemma 4.6, we can hide $6\sqrt{2m\log(2m^3)}$ processes and force the resulting configuration into one of the above categories with probability at least $1 - \frac{1}{m^3}$.

This implies that for one of the above categories, with probability at least $1 - \frac{1}{m^3}$, the vector $\vec{y}_1 \in X^C$ has a corresponding partial layer $L_{\vec{y}_1}$, such that the configuration $(C, \vec{y}_1, L_{\vec{y}_1})$ has the valence of that category. We now define the adversary σ_1 as the function that for every vector $\vec{y}_1 \in X^C$ chooses the corresponding partial layer $L_{\vec{y}_1}$, that is, $\sigma_1(\vec{y}_1) = L_{\vec{y}_1}$. Our claim is that the category that can be forced by σ_1 is the third one, that is, the resulting configuration is null-valent.

Assume, towards a contradiction, that the category that can be forced is the first one. This implies that the probability over all vectors $\vec{y}_1 \in X^C$ that $(C, \vec{y}_1, L_{\vec{y}_1})$ is $(0, k+1, S_P)$-potent is at least $1 - \frac{1}{m^3}$. Therefore, with probability at least $1 - \frac{1}{m^3}$, the vector $\vec{y}_1 \in X^C$ is such that there exists a 0-deciding adversary α' from $(C, \vec{y}_1, L_{\vec{y}_1})$ for which:

$$\Pr[\text{decision from } (C, \vec{y}_1, L_{\vec{y}_1}) \text{ under } \alpha' \text{ is } 0] > 1 - \epsilon_{k+1}$$

Therefore, with probability at least $1 - \frac{1}{m^3}$, there exists an adversary $\alpha = \sigma_1, \alpha'$ from C such that:

$$
\begin{aligned}
&\Pr[\text{decision from } C \text{ under } \alpha \text{ is } 0] \\
&\quad = \sum_{\vec{y}_1 \in X^C} \Pr[\vec{y}_1] \cdot \Pr[\text{decision from } (C, \vec{y}_1, L_{\vec{y}_1}) \text{ under } \alpha' \text{ is } 0] \\
&\quad > \left(1 - \frac{1}{m^3}\right) \cdot (1 - \epsilon_{k+1}) \\
&\quad \geq \left(1 - \frac{1}{(n-f)^3}\right) \cdot \left(1 - \frac{1}{n\sqrt{n}} + \frac{k+1}{(n-f)^3}\right) \\
&\quad = 1 - \frac{1}{n\sqrt{n}} + \frac{k}{(n-f)^3} + \frac{1}{(n-f)^3\, n\sqrt{n}} - \frac{k+1}{(n-f)^6} \\
&\quad > 1 - \frac{1}{n\sqrt{n}} + \frac{k}{(n-f)^3} = 1 - \epsilon_k,
\end{aligned}
$$

where the last inequality holds for sufficiently large n, since $(n-f)^6 \geq (n-f)^3\, n\sqrt{n}$ and $k = O(n)$. This contradicts the assumption that C is (k, S_P)-null-valent.

A similar argument holds for the second category. Hence, with probability at least $1 - \frac{1}{m^3}$, the third category can be forced, namely, we can reach a configuration that is $(k+1, S_P)$-null-valent. □

4.5. Putting the Pieces Together.

We can now put the pieces together and prove the lower bound on the total step complexity of any randomized consensus algorithm.

THEOREM 4.8. *The total step complexity of any f-tolerant randomized consensus algorithm in an asynchronous system, where $n - f \in \Omega(n)$ and $f \geq 12\sqrt{2n \log\left(2n^3\right)}$, is $\Omega(f(n-f))$.*

PROOF. We show that the probability of forcing the algorithm to continue $\frac{f}{2}$ layers is at least $1 - \frac{1}{\sqrt{n}}$. Therefore, the expected number of layers is at least $(1 - \frac{1}{\sqrt{n}}) \cdot \frac{f}{2}$. Each of these layers is an f-layer containing at least $n - f$ steps, implying that the expected total number of steps is at least $\Omega((1 - \frac{1}{\sqrt{n}}) \cdot \frac{f}{2} \cdot (n-f))$, which is in $\Omega(f(n-f))$ since $n - f \in \Omega(n)$.

We argue that for every k, $0 \leq k \leq \frac{f}{2}$, with probability at least $1 - k\frac{1}{n\sqrt{n}}$, there is a configuration C reachable by an f-execution with at least k layers, which is either v-switching or nonunivalent with respect to S_P where $|P| \leq k+1$. Once the claim is proved, the theorem follows by taking $k = \frac{f}{2}$, since the probability of having an f-execution with more than $\frac{f}{2}$ layers is at least $1 - \frac{f}{2} \cdot \frac{1}{n\sqrt{n}} > 1 - \frac{1}{\sqrt{n}}$.

We prove the claim by induction on k.

Basis: $k = 0$. By Lemma 4.1, there exists an initial configuration C that is not univalent with respect to $S_\emptyset$ or $S_{\{p\}}$, for some process p.

Induction step: Assume C is a configuration reachable by an f-execution with at least k layers, that is either v-switching or nonunivalent with respect to S_P where

$|P| \leq k+1$. We prove that with probability at least $1-\frac{1}{n\sqrt{n}}$, C can be extended with at least one layer to a configuration C' that is either v-switching or nonunivalent with respect to $S_{P'}$ where $|P'| \leq k+2$. This implies that C' exists with probability $(1-k\frac{1}{n\sqrt{n}})(1-\frac{1}{n\sqrt{n}}) \geq 1-(k+1)\frac{1}{n\sqrt{n}}$.

If C is bivalent, then by Lemma 4.3, there exists an adversary σ and a process p such that $C \circ \sigma$ is either v-switching or not $(k+1, S_P)$-univalent or not $(k+1, S_{P\cup\{p\}})$-univalent.

If C is v-switching, then by Lemma 4.4, there exists a finite adversary α_ℓ and a process p such that with probability at least $1-\frac{1}{n\sqrt{n}}$, $C \circ \alpha_\ell$ is either $\bar{v}$-switching, or not univalent with respect to S_P, or not univalent with respect to $S_{P\cup\{p\}}$.

If C is null-valent, then by Lemma 4.7, there exists an adversary σ_1 such that the configuration $C \circ \sigma_1$ is not $(k+1, S_P)$-univalent with probability at least $1-\frac{1}{m^3}$. Since $m \geq n-f \in \Omega(n)$, we have that $1-\frac{1}{m^3} \geq 1-\frac{1}{(n-f)^3} > 1-\frac{1}{n\sqrt{n}}$. □

Finally, taking $f \in \Omega(n)$ and $n-f \in \Omega(n)$, we get a lower bound of $\Omega(n^2)$ on the total step complexity.

5. *A Matching Upper Bound*

This section presents a randomized consensus algorithm with $O(n^2)$ total step complexity, by introducing a *shared coin* algorithm with a constant *agreement parameter* and $O(n^2)$ total step complexity. In a shared coin algorithm with agreement parameter δ, each process outputs a decision value -1 or $+1$, such that for every $v \in \{-1, +1\}$, there is a probability of at least δ for all processes to output the same value v [Aspnes and Herlihy 1990]. Using a shared coin algorithm with $O(n^2)$ total step complexity and a constant agreement parameter in the scheme of Aspnes and Herlihy [1990], implies a randomized consensus algorithm with $O(n^2)$ total step complexity.

As in previous shared coin algorithms [Saks et al. 1991; Bracha and Rachman 1991], in our algorithm the processes flip coins until the amount of coins that were flipped reaches a certain threshold. An array of n single-writer multi-reader registers records the number of coins each process has flipped, and their sum. A process reads the whole array in order to track the total number of coins that were flipped.

Each process decides on the value of the majority of the coin flips it reads. Our goal is for the processes to read similar sets of coins, in order to agree on the same majority value. For this to happen, we bound the total number of coins that are flipped (by any process) after some process observes that the threshold was exceeded. A very simple way to guarantee this property is to have processes frequently read the array in order to detect quickly that the threshold was reached. This, however, increases the total step complexity. Therefore, as in previous shared coin algorithms, we have to resolve the tradeoff between achieving a small total step complexity and a large (constant) agreement parameter.

The novel idea of our algorithm in order to avoid this tradeoff, is to utilize a multi-writer register called *done* that serves as a binary termination flag; it is initialized to false. A process that detects that enough coins were flipped, sets *done* to true. This allows a process to read the array only once in every n of its local coin flips, but check the register *done* before every local coin flip.

Algorithm 1 Shared coin algorithm: code for process p_i.

```
local integer $num$, initially 0
      array a[1..n]
1: while not $done$ do
2:    $num++$
3:    flip = random($-1$,$+1$)                                        // a fair local coin
4:    A[$i$].$\langle count, sum\rangle = \langle count++, sum + flip\rangle$        // atomically
5:    if $num == 0 \bmod n$ then                              // check if time to terminate
6:       $a$ = collect A                                           // $n$ read operations
7:       if $a[1].count + \cdots + a[n].count \geq n^2$ then $done$ = true
                                                                  // raise termination flag
   end while
8: $a$= collect A                                                  // $n$ read operations
9: return sign($\sum_{j=1}^{n} a[j].sum$)
               // return $+1$ or $-1$, depending on the majority value of the coin flips
```

The pseudocode appears in Algorithm 1. In addition to the binary register *done*, it uses an array A of n single-writer multi-reader registers, each with the following components (all initially 0):

count: how many flips the process performed so far.
sum: the sum of coin flips values so far.

Each process keeps a local copy a of the array A. The collect in lines 6 and 8 is an abbreviation for n read operations of the array A.

For the proof, fix an execution α of the algorithm. We will show that all processes that terminate agree on the value 1 for the shared coin with constant probability; by symmetry, the same probability holds for agreeing on -1, which implies the algorithm has a constant agreement parameter.

The total count of a specific collect is the sum of $a[1].count, \ldots, a[n].count$, as read in this collect. Note that the total count read in Line 8 is ignored, but it can still be used for the purpose of the proof.

Although we only maintain the counts and sums of coin flips, we can (externally) associate them with the set of coin flips they reflect; we denote by F_C the collection of coin flips that are written in the shared memory by the first time that true is written to *done*. The size of F_C can easily be bounded, since each process flips at most n coins before checking A.

LEMMA 5.1. *F_C contains at least n^2 coins and at most $2n^2$ coins.*

PROOF. Clearly, true is written to *done* in line 7 only if the process reads at least n^2 flips, therefore $|F_C| \geq n^2$. Consider the point in the execution after n^2 coins were written. Each process can flip at most n more coins until reaching line 7, and then it writes true to *done*. Therefore when true is written to *done* there are at most $2n^2$ coins written, and $|F_C| \leq 2n^2$. □

For a set of coins F we let $Sum(F)$ be the sum of the coins in F. We denote by F_i the set of coin flips read by the collect of process p_i in Line 8. This is the set according to which the process p_i decides on its output, that is, p_i returns $Sum(F_i)$. Since each process may flip at most one more coin after true is written to *done*, we can show:

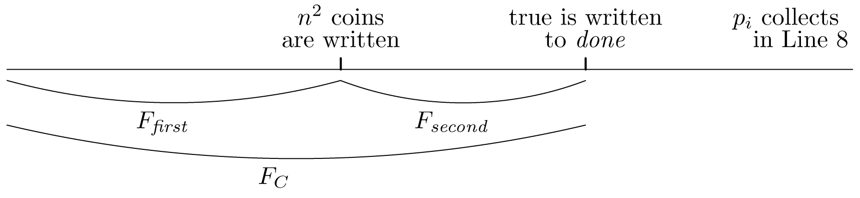

FIG. 4. Phases of the shared coin algorithm.

LEMMA 5.2. *For every i, $F_C \subseteq F_i$, and $F_i \backslash F_C$ contains at most $n - 1$ coins.*

PROOF. Note that the collect of any process p_i in Line 8 starts after true is written to *done*. Hence, F_i contains F_C.

After true is written to *done*, each process (except the process that had written true to *done*) can flip at most one more coin before reading that *done* is true in Line 1. Therefore, the set of coins that are written when the process reads Line 8 is the set of coins that were written when true was written to *done* (which is F_C), plus at most $n - 1$ additional coins. $\square$

We now show that there is at least a constant probability that $Sum(F_C) \geq n$. In this case, by Lemma 5.2, all processes that terminate agree on the value 1, since F_i contains at most $n - 1$ additional coins.

We partition the execution into three phases. The first phase ends when n^2 coins are written. After every coin is written there may be up to $n - 1$ additional coins that are already flipped but not yet written. The adversary has a choice whether to allow each of these coins to be written. We assume an even stronger adversary that can choose the n^2 written coins out of $n^2 + n - 1$ coins that were flipped. The second phase ends when true is written to *done*. In the third phase, each process reads the whole array A and returns a value for the shared coin. (See Figure 4.)

Since F_C is the set of coins written when *done* is set to true, then it is exactly the set of coins written in the first and second phases. Let F_{first} be the first n^2 coins that are written, and $F_{second} = F_C \backslash F_{first}$. This implies that $Sum(F_C) = Sum(F_{first}) + Sum(F_{second})$. Therefore, we can bound (from below) the probability that $Sum(F_C) \geq n$ by bounding the probabilities that $Sum(F_{first}) \geq 3n$ and $Sum(F_{second}) \geq -2n$.

Consider the sum of the first $n^2 + n - 1$ coin flips. After these coins are flipped, the adversary has to write at least n^2 of them, which will be the coins in F_{first}. If the sum of the first $n^2 + n - 1$ coin flips is at least $4n$ then $Sum(F_{first}) \geq 3n$. We bound the probability that this happens using the Central Limit Theorem.

LEMMA 5.3. *The probability that $Sum(F_{first}) \geq 3n$ is at least $\frac{1}{8\sqrt{2\pi}}e^{-8}$.*

PROOF. There are $N = n^2 + n - 1$ coins flipped when n^2 coins are written to F_{first}. By the Central Limit Theorem, the probability for the sum of these coins to be at least $x\sqrt{N}$, converges to $1 - \Phi(x)$, where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy$ is the normal distribution function. By Feller [1968, Chapter VII], we have $1 - \Phi(x) > (\frac{1}{x} - \frac{1}{x^3})\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$. Substituting $x = 4$ we have that with probability at least $\frac{1}{8\sqrt{2\pi}}e^{-8}$ the sum of these N coins is at least $4\sqrt{N}$, which is more than $4n$, and hence $Sum(F_{first}) \geq 3n$. $\square$

We now need to bound $Sum(F_{second})$. Unlike F_{first}, whose size is determined, the adversary may have control over the number of coins in F_{second}, and not only over which coins are in it. However, by Lemma 5.1 we have $|F_C| \leq 2n^2$, therefore $|F_{second}| \leq n^2$, which implies that F_{second} must be some prefix of n^2 additional coin flips. We consider the partial sums of these n^2 additional coin flips, and show that with high probability, all these partial sums are greater than $-2n$, and therefore in particular $Sum(F_{second}) > -2n$.

Formally, for every i, $1 \leq i \leq n^2$, let X_i be the ith additional coin flip, and denote $S_j = \sum_{i=1}^{j} X_i$. Since $|F_{second}| \leq n^2$, there exist k, $1 \leq k \leq n^2$, such that $S_k = Sum(F_{second})$. If $S_j > -2n$ for every j, $1 \leq j \leq n^2$, then, specifically $Sum(F_{second}) = S_k > -2n$.

The bound on the partial sums is derived using Kolmogorov's inequality.

KOLMOGOROV'S INEQUALITY [FELLER 1968, CHAPTER IX]. *Let $X_1, \ldots, X_m$ be independent random variables such that $Var[X_i] < \infty$ for every i, $1 \leq i \leq m$, and let $S_j = \sum_{i=1}^{j} X_i$ for every j, $1 \leq j \leq m$. Then, for every $\lambda > 0$, the probability that*

$$|S_j - E[S_j]| < \lambda\sqrt{Var[S_m]}\ , \textit{ for all } j,\ 1 \leq j \leq m,$$

is at least $1 - \lambda^{-2}$.

LEMMA 5.4. *The probability that $S_j > -2n$ for all j, $1 \leq j \leq n^2$, is at least $\frac{3}{4}$.*

PROOF. The results of the n^2 coin flips are independent random variables $X_1, \ldots, X_{n^2}$, with $E[X_i] = 0$ and $Var[X_i] = 1$, for every i, $1 \leq i \leq n^2$.

Since S_j is the sum of j independent random variables, its expectation is $E[S_j] = \sum_{i=1}^{j} E[X_i] = 0$, and its variance is $Var[S_j] = \sum_{i=1}^{j} Var[X_i] = j$.

Kolmogorov's inequality implies that $|S_j| < 2n$, for all j, $1 \leq j \leq n^2$, with probability at least $\frac{3}{4}$. □

This bounds the probability of agreeing on the same value for the shared coin as follows:

LEMMA 5.5. *Algorithm 1 is a shared coin algorithm with agreement parameter* $\delta = \frac{3}{32\sqrt{2\pi}}e^{-8}$.

PROOF. By Lemma 5.3, the probability that $Sum(F_{first}) \geq 3n$ is at least $\frac{1}{8\sqrt{2\pi}}e^{-8}$, and by Lemma 5.4, the probability that $Sum(F_{second}) \geq -2n$ is at least $\frac{3}{4}$. Since $Sum(F_C) = Sum(F_{first}) + Sum(F_{second})$, this implies that the probability that $Sum(F_C) \geq n$ is at least $\frac{3}{32\sqrt{2\pi}}e^{-8}$.

By Lemma 5.2, for every i, $F_i \backslash F_C$ contains at most $n - 1$ coins, which implies that if $Sum(F_C) \geq n$ then $Sum(F_i) \geq 1$, and therefore if p_i terminates, then it will decide 1. Hence, with probability at least $\frac{3}{32\sqrt{2\pi}}e^{-8}$, $Sum(F_C) \geq n$ and all processes which terminate agree on the value 1.

By symmetry, all processes which terminate agree on the value -1 with at least the same probability. □

Clearly, Algorithm 1 flips $O(n^2)$ coins. Moreover, all work performed by processes in reading the array A can be attributed to coin flips. This can be used to show that Algorithm 1 has $O(n^2)$ total step complexity.

LEMMA 5.6. *Algorithm 1 has $O(n^2)$ total step complexity.*

PROOF. We begin by counting operations that are not part of a collect. There are $O(1)$ such operations per local coin flip, and by Lemmas 5.1 and 5.2 there are at most $2n^2 + n - 1$ local coin flips, implying $O(n^2)$ operations that are not part of a collect.

Each collect performed by p_i in Line 6, can be associated with the n local coins that can be flipped by p_i before it. By Lemma 5.1, there are at most $2n^2$ coins in F_C, that is, at most $2n^2$ coin flips during the first and second phases of the algorithm. Therefore, during these phases there can be at most $\frac{2n^2}{n} = 2n$ collects performed by processes in Line 6. In the third phase, every process may perform another collect in Line 6, and another collect in Line 8, yielding at most $2n$ additional collects. Thus, there are at most $4n$ collects performed during the algorithm, yielding a total of $O(n^2)$ steps. □

Using Algorithm 1 in the scheme of Aspnes and Herlihy [1990] yields a randomized consensus algorithm. Informally, their scheme uses single-writer registers in which the processes update their preferences (starting with their inputs), and allows the processes to collect the values of these registers in order to detect agreement. If fast processes agree then they decide their preference and a slow process also adopts it, otherwise a process changes its preference according to the result of the shared coin. Given a shared coin algorithm with agreement parameter δ and step complexity T, the scheme yields a randomized consensus algorithm with $O(\delta^{-1}T)$ expected step complexity. Since our shared coin algorithm has a constant agreement parameter and $O(n^2)$ step complexity, we get the next theorem:

THEOREM 5.7. *There is a randomized consensus algorithm with $O(n^2)$ total step complexity.*

6. *Discussion*

We have shown that $\Theta(n^2)$ is a tight bound on the total step complexity of solving randomized consensus, under a strong adversary, in an asynchronous shared-memory system using multi-writer registers.

Aspnes [1998] shows an $\Omega(\frac{n^2}{\log^2 n})$ lower bound on the expected total number of coin flips. Our layering approach as presented here, does not distinguish between a deterministic step and a step involving a coin flip, leaving open the question of the optimal number of coin flips.

Our algorithm exploits the multi-writing capabilities of the register. The best known randomized consensus algorithm using single-writer registers [Bracha and Rachman 1991] has $O(n^2 \log n)$ total step complexity, and it is intriguing to close the gap from our lower bound.

In addition to settling the open question of the asymptotic total step complexity, we consider an important contribution of this article to be in introducing new techniques for investigating randomized consensus algorithms. These techniques provide a new context and opportunities for exploring several research directions, two of which are discussed next.

One direction is to study the *individual* step complexity of randomized consensus; namely, the expected number of steps performed by a single process. Aspnes and

Waarts [1996] present a shared coin algorithm in which each process performs $O(n \log^2 n)$ expected number of steps; their algorithm has $O(n^2 \log^2 n)$ total step complexity. Their shared coin algorithm is similar to Bracha and Rachman [1991], but the local flips of a process are assigned increasing weights, which allows fast processes to terminate earlier. In a recent work [Aspnes et al. 2008], we have shown that the individual step complexity can be reduced to $O(n \log n)$, using a multi-writer register; reducing the individual step complexity further, or avoiding the use of a multi-writer register, remain open questions.

Our results may also provide new insight into the expected step complexity of randomized consensus under *weak* adversaries that cannot observe the local flips or the contents of the shared memory. Several papers presented randomized consensus algorithms for this type of adversaries, (e.g., Aumann and Bender [2005], Aumann [1997], Chandra [1996], Chor et al. [1987], and Aumann and Kapah-Levy [1999]), with the best algorithms having $O(n \text{ polylog}(n))$ total step complexity and $O(\text{polylog}(n))$ individual step complexity.

Weaker adversaries have also been studied in message passing systems. In the *full information* model, the adversary is computationally unbounded and has access to all messages, but not to the processes' random bits. Furthermore, the adversary is nonadaptive, that is, chooses the faulty processes before the algorithm begins. Several papers develop algorithms for Byzantine Agreement or Leader Election in the full information model [Ben-Or et al. 2006; Goldwasser et al. 2006; King et al. 2006a; King et al. 2006b; Kapron et al. 2008]. Our lower bound on the expected total step does not apply to this model since we assume a much stronger adversary. Recently [Attiya and Censor 2008], we have applied techniques used to show potence connectivity in Section 2 to bound the termination probability of bounded consensus algorithms, under a very weak adversary.

ACKNOWLEDGMENTS. We would like to thank Faith Ellen, Javier Garcia, David Hay, Danny Hendler, Sergio Rajsbaum and an anonymous referee for insightful comments on earlier versions of this article.

REFERENCES

ABRAHAMSON, K. 1988. On achieving consensus using a shared memory. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 291–302.

ALON, N., AND SPENCER, J. H. 2000. *The Probabilistic Method*, 2nd ed. Wiley, New York.

ASPNES, J. 1990. Time- and space-efficient randomized consensus. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 325–332.

ASPNES, J. 1998. Lower bounds for distributed coin-flipping and randomized consensus. *J. ACM 45*, 3 (May), 415–450.

ASPNES, J. 2003. Randomized protocols for aynchronous consensus. *Distrib. Comput. 16*, 2-3 (Sept.), 165–176.

ASPNES, J., ATTIYA, H., AND CENSOR, K. 2008. Randomized consensus in expected $O(n \log n)$ individual work. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 325–334.

ASPNES, J., AND HERLIHY, M. 1990. Fast randomized consensus using shared memory. *J. Algorithms 11*, 3, 441–461.

ASPNES, J., AND WAARTS, O. 1996. Randomized consensus in expected $O(n \log^2 n)$ operations per processor. *SIAM J. Comput. 25*, 5, 1024–1044.

ATTIYA, H., AND CENSOR, K. 2008. Lower bounds for randomized consensus under a weak adversary. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 315–324.

ATTIYA, H., DOLEV, D., AND SHAVIT, N. 1989. Bounded polynomial randomized consensus. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 281–293.

ATTIYA, H., AND WELCH, J. L. 2004. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, 2nd ed. Wiley, New York.

AUMANN, Y. 1997. Efficient asynchronous consensus with the weak adversary scheduler. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 209–218.

AUMANN, Y., AND BENDER, M. A. 2005. Efficient low-contention asynchronous consensus with the value-oblivious adversary scheduler. *Distrib. Comput. 17*, 3 (Mar.), 191–207.

AUMANN, Y., AND KAPAH-LEVY, A. 1999. Cooperative sharing and asynchronous consensus using single-reader single-writer registers. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, New York, 61–70.

BAR-JOSEPH, Z., AND BEN-OR, M. 1998. A tight lower bound for randomized synchronous consensus. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 193–199.

BEN-OR, M., PAVLOV, E., AND VAIKUNTANATHAN, V. 2006. Byzantine agreement in the full-information model in o(log n) rounds. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, 179–186.

BRACHA, G., AND RACHMAN, O. 1991. Randomized consensus in expected $O(n^2 \log n)$ operations. In *Proceedings of the 5th International Workshop on Distributed Algorithms (WDAG)*. Springer-Verlag, Berlin/Heidelberg, Germany, 143–150.

CHANDRA, T. D. 1996. Polylog randomized wait-free consensus. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 166–175.

CHOR, B., ISRAELI, A., AND LI, M. 1987. On processor coordination using asynchronous hardware. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, 86–97.

DOLEV, D., AND STRONG, H. R. 1983. Authenticated algorithms for byzantine agreement. *SIAM J. Comput. 12*, 4, 656–666.

FELLER, W. 1968. *An Introduction to Probability Theory and Its Applications*, 3rd ed. Vol. 1. Wiley, New York.

FISCHER, M. J., AND LYNCH, N. A. 1982. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett. 14*, 4, 183–186.

FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. 1985. Impossibility of distributed consensus with one faulty process. *J. ACM 32*, 2 (Apr.), 374–382.

GOLDWASSER, S., PAVLOV, E., AND VAIKUNTANATHAN, V. 2006. Fault-tolerant distributed computing in full-information networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (Washington, DC) IEEE Computer Society Press, Los Alamitos, CA, 15–26.

HERLIHY, M. 1991. Wait-free synchronization. *ACM Trans. Program. Lang. Syst. 13*, 1 (Jan.), 124–149.

KAPRON, B., KEMPE, D., KING, V., SAIA, J., AND SANWALANI, V. 2008. Fast asynchronous byzantine agreement and leader election with full information. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM, New York, 1038–1047.

KING, V., SAIA, J., SANWALANI, V., AND VEE, E. 2006a. Scalable leader election. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*. ACM, New York, 990–999.

KING, V., SAIA, J., SANWALANI, V., AND VEE, E. 2006b. Towards secure and scalable computation in peer-to-peer networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (Washington, DC) IEEE Computer Society Press, Los Alamitos, CA, 87–98.

LAMPORT, L. 1998. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (May), 133–169.

LOUI, M. C., AND ABU-AMARA, H. H. 1987. *Memory Requirements for Agreement Among Unreliable Asynchronous Processes*. JAI Press, Greenwich, Conn., 163–183.

LYNCH, N. A. 1996. *Distributed Algorithms*. Morgan-Kaufmann, San Mateo, CA.

MOSES, Y., AND RAJSBAUM, S. 2002. A layered analysis of consensus. *SIAM J. Comput. 31*, 4, 989–1021.

MOTWANI, R., AND RAGHAVAN, P. 1995. *Randomized Algorithms*. Cambrigde University Press, Cambridge, UK.

RAYNAL, M., AND ROY, M. 2005. A note on a simple equivalence between round-based synchronous and asynchronous models. In *Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing (PRDC)* (Washington, DC) IEEE Computer Society Press, Los Alamitos, CA, 387–392.

SAKS, M., SHAVIT, N., AND WOLL, H. 1991. Optimal time randomized consensus—making resilient algorithms fast in practice. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, New York, 351–362.

SANTORO, N., AND WIDMAYER, P. 1989. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*. Springer-Verlag, Berlin/Heidelberg, Germany 304–313.

SCHECHTMAN, G. 1981. Levy type inequality for a class of finitie metric spaces. In *Lecture Notes in Mathematics: Martingle Theory in Harmonic Analysis and Banach Spaces*. Vol. 939. Springer-Verlag, Berlin/Heidelberg, Germany 211–215.

SCHNEIDER, F. B. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv. 22*, 4 (Dec.), 299–319.

RECEIVED OCTOBER 2007; ACCEPTED JULY 2008