

# Lower Bounds for Randomized Consensus under a Weak Adversary

Hagit Attiya\*  
Department of Computer  
Science, Technion  
Haifa, Israel  
hagit@cs.technion.ac.il

Keren Censor\*†  
Department of Computer  
Science, Technion  
Haifa, Israel  
ckeren@cs.technion.ac.il

## ABSTRACT

This paper studies the inherent trade-off between termination probability and total step complexity of randomized consensus algorithms. It shows that for every integer  $k$ , the probability that an  $f$ -resilient randomized consensus algorithm of  $n$  processes does not terminate with agreement within  $k(n - f)$  steps is at least  $\frac{1}{c^k}$ , for some constant  $c$ .

The lower bound holds for asynchronous systems, where processes communicate either by message passing or through shared memory, under a very weak adversary that determines the schedule in advance, without observing the algorithm's actions. This complements algorithms of Kapron et al. [22], for message-passing systems, and of Aumann et al. [6, 7], for shared-memory systems.

## Categories and Subject Descriptors

D.1.3 [Software]: Programming Techniques—*Concurrent programming*; F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; G.3 [Mathematics of Computing]: Probability and Statistics

## General Terms

Algorithms, Theory

## Keywords

distributed computing, shared memory, message passing, lower bound, randomized algorithms, consensus

## 1. INTRODUCTION

At the heart of many coordination problems in distributed systems lies the need to reach *consensus* among processes,

\*Supported in part by the *Israel Science Foundation* (grant number 953/06).

†Supported in part by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.  
Copyright 2008 ACM 978-1-59593-989-0/08/08 ...\$5.00.

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*), and every process must eventually decide (*termination*). It is well-known that no deterministic algorithm can achieve consensus in an *asynchronous* system, if one process may fail [19, 21, 26]. One successful approach for circumventing this impossibility result is to employ randomization, and relax the termination property to hold with high probability. In typical randomized algorithms for consensus, the probability of *not* terminating in agreement decreases as the execution progresses, becoming smaller as processes perform more steps.

This paper shows that this behavior is inherent, by proving lower bounds on the termination probability when the step complexity is bounded. In order to make the lower bounds as strong as possible, we assume a very weak adversarial model, which fixes the complete schedule in advance, without observing the steps of the algorithm. In particular, the schedule is determined without knowing results of local coin-flips, contents of messages sent, or memory locations accessed.

We prove that for every integer  $k$ , the probability that an  $f$ -resilient randomized consensus algorithm of  $n$  processes does not terminate after  $k(n - f)$  steps is at least  $\frac{1}{c^k}$ , where  $c$  is a constant if  $\lceil \frac{n}{f} \rceil$  is a constant. The result holds for asynchronous message-passing systems and asynchronous shared-memory systems (using reads and writes), albeit with different constants. While the same general proof structure applies in both cases, it is accomplished differently in the message-passing and the shared-memory models; the latter case is further complicated due to the adversary's weakness.

For the message-passing model, our proof extends and improves on a result of Chor, Merritt and Shmoys [16] for *synchronous* message-passing systems. They show that the probability that a randomized consensus algorithm does not terminate after  $k$  rounds (and  $k(n - f)$  steps) is at least  $\frac{1}{c \cdot k^k}$ . (A similar result is attributed to Karlin and Yao [23].) The proof rests on considering a specific chain of *indistinguishable* executions and showing a correlation between the termination probability and the length of this chain,<sup>1</sup> which in turn depends on the number of rounds. The chain is taken from the proof of the rounds lower bound for (deterministic) consensus [17, 18] (cf. [5, Chapter 5]); since the chain is determined in advance, i.e., regardless of the algorithm's transitions, the lower bound is derived with a weak

<sup>1</sup>The *length* of the chain is the number of executions in it.

adversary. (An overview of the proof strategy appears in Section 2.2.)

Our first contribution, for the message-passing model, improves on this lower bound by exploiting the fact that asynchrony allows to construct “shorter” indistinguishability chains. This shows that the probability that an asynchronous randomized consensus algorithm does not terminate after  $k(n - f)$  steps is at least  $\frac{1}{c^k}$ , where  $c$  is a constant if  $\lceil \frac{n}{f} \rceil$  is a constant. (The lower bound for asynchronous message-passing systems appears in Section 3.)

Substituting specific values in our lower bound implies that any randomized consensus algorithm has probability at least  $\frac{1}{\text{poly}(\log(n))}$  for not terminating within  $\log \log n$  (asynchronous) rounds, and probability at least  $\frac{1}{\text{poly}(n)}$  for not terminating within  $\log n$  (asynchronous) rounds. These lower bounds are tight due to a recent consensus algorithm of Kapron et al. [22], which terminates in agreement within a polylogarithmic number of (asynchronous) rounds with probability  $1 - \frac{1}{\text{poly}(\log(n))}$ .

There is an extensive literature on randomized agreement algorithms for message passing systems. Recent papers in this area provide algorithms for agreement in the presence of Byzantine processes in *full information* models, where the adversary is computationally unbounded. See [10, 20, 24] for a more detailed description and references.

In principle, the lower bound scheme can be extended to the shared-memory model by focusing on *layered* executions [4, 27]. However, our severely handicapped adversarial model poses a couple of technical challenges. First, while in the message-passing model each step can be assumed to send messages to all processes, in a shared-memory event, a process chooses which register to access and whether to read from it or write to it. A very weak adversary, as we use for our lower bounds, must find a way to make its scheduling decisions in advance without even knowing what type of step the process will take. Second, the proof scheme requires schedules to be determined independently of the coin-flips. The latter difficulty cannot be alleviated even by assuming a stronger adaptive adversary that may schedule the processes according to the execution so far.

We manage to extend the lower bound scheme to the shared-memory model, by first simplifying the model, assuming that processes either write to *single-writer* registers or perform a *cheap snapshot* operation, reading all the registers at once. By further assuming that an algorithm regularly alternates between writes and cheap snapshots, we make processes’ steps predictable, allowing a weak adversary to construct indistinguishability chains. The lower bound is extended to hold for multi-writer registers by reduction; while ordinary simulations of multi-writer registers using single-writer registers have  $O(n)$  overhead (which would nullify the lower bound), cheap snapshots admit a simulation with constant overhead. (The results for the shared-memory model appear in Section 4.)

The lower bounds we obtain for the shared-memory model are the same as for the message-passing model, though with different constants. (More detailed calculations and comparison with related work appear in Section 6.)

To the best of our knowledge, there are no other lower bounds on randomized consensus in shared-memory systems under a weak adversary. There are several algorithms assuming a *value-oblivious* adversary, which may determine

the schedule adaptively based on the functional description of past and pending operations, but cannot observe any value of any register nor results of local coin-flips. This model is clearly stronger than the adversary we employ, and hence our lower bounds apply to it as well.

The algorithms differ by the type of shared registers they use [6–8, 12]. For single-writer multi-reader registers, Aumann and Bender [7] give a consensus algorithm that has probability of at most  $\frac{1}{n^c}$  of not terminating within  $O(n \log^2 n)$  steps. For multi-writer multi-reader registers, Aumann [6] shows a consensus algorithm in which the probability of not terminating in  $k$  iterations (and  $O(k \cdot n)$  steps) is at most  $(3/4)^k$ .

Chandra [12] gives an algorithm with  $O(\log^2 n)$  *individual* step complexity, assuming an intermediate adversary that cannot see the outcome of a coin-flip until it is read by some process. Aumann and Kapah-Levy [8] give an algorithm with  $O(n \log n \exp(2\sqrt{\text{poly}(\log n)}))$  total step complexity, using single-writer single-reader registers, and assuming a value-oblivious adversary.

An algorithm with  $O(n \log \log n)$  total step complexity against a weak adversary was given by Cheung [14], which considers a model with a stronger assumption that a write operation occurs atomically after a local coin-flip. It improves upon earlier work by Chor, Israeli, and Li [15], who provide an algorithm with  $O(n^2)$  total step complexity using a slightly different atomicity assumption.

Other related work on randomized consensus assumes strong adversaries, which adapt to the computation, scheduling processes dynamically, after observing the results of their local coin-flips. A lot of study was invested, yielding numerous algorithms (see the survey in [2]). Lower bounds were given on the expected number of coin-flips [1], on the expected number of rounds in synchronous systems [9], and a tight  $\Theta(n^2)$  bound on the total step complexity in asynchronous systems [4].

## 2. THE LOWER BOUND STRATEGY

### 2.1 The Model in Brief

We consider a standard model of an asynchronous system with a set of  $n$  processes  $P = \{p_1, \dots, p_n\}$ . Each *step* of a process consists of some local computation, including an arbitrary number of coin-flips (possibly biased), and a communication operation, which depends on the communication model.

In a message passing system processes communicate by sending and receiving messages: the communication operation of a process is sending messages to some subset of the processes, and receiving messages from some subset of them. For the lower bounds, we assume that a process sends a message to all the processes in each step. In a shared memory system processes communicate by reading and writing to shared registers; each step of a process is either a read or a write to some register.

The local coin-flips of a process  $p_i$  are modelled as a string of (possibly biased) coin-flips  $c_i$ , which is unavailable to any other process. All of the randomization of the algorithm is encompassed within  $n$  coin-flip strings  $c = (c_1, \dots, c_n)$ .

Additional non-determinism is introduced by the scheduling choices made by an *adversary*. We assume a *weak* adversary that is non adaptive and decides on the scheduling in advance. The adversary does not observe the results of

any local coins a process flips, nor any operation a process performs.

A schedule  $\sigma$ , together with an initial configuration  $I$  and  $n$  coin-flip strings  $c = (c_1, \dots, c_n)$ , determine an *execution*  $\alpha(\sigma, c, I)$ .

## 2.2 The Lower Bound Approach

Let  $A$  be an  $f$ -resilient asynchronous randomized consensus algorithm. Let  $q_k$  denote the maximum probability, over all weak adversaries and over all initial configurations, that  $A$  does not terminate after a total of  $k(n-f)$  steps are taken.

In order to prove a lower bound on  $q_k$ , we consider a restricted set of schedules that proceed in layers [4, 27]. 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.

We will consider only schedules that are  $f$ -schedules. A schedule  $\sigma = L_1, L_2, \dots, L_k$  is an  $f$ -schedule, if it is a finite sequence of  $f$ -layers. A process  $p_i$  is *non-faulty* in layer  $r$  if it appears in the layer. A process  $p_i$  *crashes* in layer  $r$  if it does not take a step in any layer  $\ell \geq r$ . A process is *faulty* in layer  $r$  but does not crash in layer  $r$ , if it appears in any of the following layers.

**DEFINITION 1.** For a schedule  $\sigma$ , let  $\text{crash}(\sigma, p, r)$  be the schedule that is the same as  $\sigma$ , except that  $p$  crashes in layer  $r$ , i.e., does not take a step in any layer  $\ell \geq r$ . For a set  $P$  of processes,  $\text{crash}(\sigma, P, r)$  is defined similarly.

As mentioned in the introduction, our proof will make use of indistinguishable executions. Intuitively, two finite executions  $\alpha$  and  $\alpha'$  are indistinguishable to a process  $p$  if it cannot tell the difference between them. This implies that  $p$  terminates in  $\alpha$  with a decision value  $v$  if and only if it terminates in  $\alpha'$  with a decision value  $v$ . The formal definition of indistinguishability is model-dependent and will be given separately in Sections 3 and 4, but we proceed formally to define indistinguishability chains, as follows.

Given two executions  $\alpha_1$  and  $\alpha_2$  with the same  $n$  coin-flip strings  $c = (c_1, \dots, c_n)$ , we denote  $\alpha_1 \stackrel{p_i}{\approx} \alpha_2$  if process  $p_i$  does not distinguish between  $\alpha_1$  and  $\alpha_2$ , and does not crash in them. In this case,  $p_i$  decides on the same value in  $\alpha_1$  and in  $\alpha_2$ . We denote  $\alpha_1 \approx_m \alpha_2$  if there is a chain of executions  $\beta_1, \dots, \beta_{m+1}$  such that

$$\alpha_1 = \beta_1 \stackrel{p_{i_1}}{\approx} \beta_2 \dots \stackrel{p_{i_m}}{\approx} \beta_{m+1} = \alpha_2 \quad .$$

We call such a chain an *indistinguishability chain*. Clearly, if  $\alpha \approx_m \beta \approx_{m'} \gamma$  then  $\alpha \approx_{m+m'} \gamma$ , for every pair of integers  $m$  and  $m'$ .

For every pair of consecutive executions in the chain, there is a process that decides on the same value in both executions. By the agreement condition, the decision in  $\alpha_1$  and in  $\alpha_2$  must be the same. This is the main idea of the lower bound proof, which is captured in Theorem 1: we take two executions that must have different agreement values and construct an indistinguishability chain between them, which bounds the probability of terminating in terms of the length of the chain. Two such executions exist by the validity condition, and we formalize them as follows.

We partition the processes into  $S = \max\{3, \lceil \frac{n}{f} \rceil\}$  sets  $P_1, \dots, P_S$ , each with at most  $f$  processes. For example, if  $n > 2f$ ,  $P_i = \{p_{(i-1)f+1}, \dots, p_{i \cdot f}\}$  for every  $i$ ,  $1 \leq i < S$ , and  $P_S = \{p_{(S-1)f+1}, \dots, p_n\}$ .

Consider initial configurations  $C_0, \dots, C_S$ , such that in  $C_0$  all the inputs are 0, and in  $C_i$ ,  $1 \leq i \leq S$ , all processes in  $P_1, \dots, P_i$  have input 1 and all other processes have input 0; in particular, in  $C_S$  all processes have input 1.

Let  $\sigma_{full}$  be the full synchronous schedule with  $k$  layers, in which no process fails. The following theorem is the main tool for bounding  $q_k$  as a function of  $m$ , the length of an indistinguishability chain. This theorem distills the technique we borrow from [16].

**THEOREM 1.** Assume there is an integer  $m$  such that for any sequences of coins  $c$ ,  $\alpha(\sigma_{full}, c, C_0) \approx_m \alpha(\sigma_{full}, c, C_S)$ . Then the probability that  $A$  does not terminate after  $k(n-f)$  steps is  $q_k \geq \frac{1}{m+1}$ .

**PROOF.** Assume, by way of contradiction, that  $q_k(m+1) < 1$ . Since  $\alpha(\sigma_{full}, c, C_0) \approx_m \alpha(\sigma_{full}, c, C_S)$ , there is a chain of  $m+1$  executions,

$$\alpha(\sigma_{full}, c, C_0) = \beta_1 \stackrel{p_{i_1}}{\approx} \beta_2 \dots \stackrel{p_{i_m}}{\approx} \beta_{m+1} = \alpha(\sigma_{full}, c, C_S) \quad .$$

The probability that  $A$  does not terminate in at least one of these  $m+1$  executions is at most  $q_k(m+1)$ . By assumption,  $q_k(m+1) < 1$ , and hence, the set  $B$  of sequences of coins  $c$  such that  $A$  terminates in all  $m+1$  executions has probability  $\Pr[c \in B] > 0$ . Since  $\alpha(\sigma_{full}, c, C_0) \approx_m \alpha(\sigma_{full}, c, C_S)$ , the agreement condition implies that the decision in all  $m+1$  executions is the same. However, the validity condition implies that the decision in  $\alpha(\sigma_{full}, c, C_0)$  is 0, and the decision in  $\alpha(\sigma_{full}, c, C_S)$  is 1, which is a contradiction.  $\square$

A slight extension of the above theorem allows us to handle *Monte-Carlo* algorithms, where processes may terminate without agreement with some small probability  $\epsilon$ . This extension is presented in Section 5.

The statement of Theorem 1 indicates that our goal is to show the existence of an integer  $m$  such that  $\alpha(\sigma_{full}, c, C_0) \approx_m \alpha(\sigma_{full}, c, C_S)$ ; clearly, the smaller  $m$ , the higher the lower bound. The following lemma comes in handy when we construct these chains.

**LEMMA 2.** Assume there is an integer  $m$  such that for every schedule  $\sigma$ , initial configuration  $I$ , sequence of coins  $c$  and set  $P_i$ ,  $\alpha(\sigma, c, I) \approx_m \alpha(\text{crash}(\sigma, P_i, 1), c, I)$ . Then  $\alpha(\sigma_{full}, c, C_0) \approx_{S(2m+1)} \alpha(\sigma_{full}, c, C_S)$ , for all sequences of coins  $c$ .

**PROOF.** Consider the schedules  $\sigma_0 = \sigma_{full}$ , and  $\sigma_i = \text{crash}(\sigma_0, P_i, 1)$  for every  $i$ ,  $1 \leq i \leq S$ , and the corresponding executions  $\alpha_{i,j} = \alpha(\sigma_i, c, C_j)$  for every  $i$  and  $j$ ,  $1 \leq i \leq S$  and  $0 \leq j \leq S$  (the execution  $\alpha_{i,j}$  starts from the initial configuration  $C_j$  with a schedule which is almost full, except that processes in  $P_i$  never take any steps).

By assumption,  $\alpha_{0,j} \approx_m \alpha_{i,j}$  for every  $i$ ,  $1 \leq i \leq S$ , and every  $j$ ,  $0 \leq j \leq S$ . Since processes in  $P_i$  are crashed in  $\sigma_i$  for every  $i$ ,  $1 \leq i \leq S$ , we have that  $\alpha_{i,i-1} \stackrel{p}{\approx} \alpha_{i,i}$ , for every process  $p \in P \setminus P_i$ . This implies that  $\alpha_{i,i-1} \approx_1 \alpha_{i,i}$ , for every  $i$ ,  $1 \leq i \leq S$ . Thus,

$$\begin{aligned} \alpha(\sigma_{full}, c, C_0) &= \alpha_{0,0} \approx_m \alpha_{1,0} \approx_1 \alpha_{1,1} \approx_m \alpha_{0,1} \\ &\approx_m \alpha_{2,1} \approx_1 \alpha_{2,2} \dots \alpha_{S,S} \approx_m \alpha_{0,S} = \alpha(\sigma_{full}, c, C_S) \quad . \end{aligned}$$

Therefore,  $\alpha(\sigma_{full}, c, C_0) \approx_{S(2m+1)} \alpha(\sigma_{full}, c, C_S)$ .  $\square$

### 3. TRADEOFF FOR THE MESSAGE-PASSING MODEL

In this section we derive the lower bound for the message-passing model. Notice that in the message-passing model, since a step consists of both sending and receiving messages, a layer  $L$  is not only a set of processes, but also specifies for each process  $p \in L$  the set of processes it receives a message from (recall that we assumed that it sends messages to all processes). The reception of messages in a certain layer is done after all messages of that layer are sent, and therefore the order of processes in a layer is insignificant.

Formally, an  $f$ -layer is a sequence  $p_{i_1}, \dots, p_{i_m}$  of distinct process id's, followed by a sequence  $M_{i_1}, \dots, M_{i_m}$  of subsets of process id's, where  $M_{i_j}$  is the subset of process id's appearing in the layer from which  $p_{i_j}$  receives a message in this layer.

Recall that the processes are partitioned into  $S = \max\{3, \lceil \frac{n}{f} \rceil\}$  sets  $P_1, \dots, P_S$ , each with at most  $f$  processes. We manipulate schedules in order to delay messages, as follows.

**DEFINITION 2.** *Let  $\sigma$  be a schedule. Let  $\text{delay}(\sigma, P_i, P_j, r)$  be the schedule that is the same as  $\sigma$ , except that the messages sent by processes in  $P_i$  in layer  $r$  are received by processes in  $P_j$  only after the  $k$ -th layer (the messages are effectively "erased" from the execution). More formally, if  $M_p$  is the subset of processes that a process  $p$  receives a message from in layer  $r$  in  $\sigma$ , then for every process  $p \in P_j$  the subset of processes that it receives a message from in layer  $r$  in  $\text{delay}(\sigma, P_i, P_j, r)$  is  $M_p \setminus P_i$ .*

We define indistinguishability of executions in the message-passing model as follows: two executions are indistinguishable to process  $p$  if it has the same local states throughout both executions. More specifically, in both executions  $p$  sends and receives the same messages, in the same order.

Clearly, at the end of layer  $r$ , any process not in  $P_j$  does not distinguish between the execution so far of a schedule  $\sigma$  and an execution so far of  $\text{delay}(\sigma, P_i, P_j, r)$ . Therefore we have:

**LEMMA 3.** *Let  $\sigma$  be a schedule with  $k$  layers. For any sequences of coins  $c$ , and initial configuration  $I$ , at the end of layer  $r$  only processes in  $P_j$  distinguish between  $\alpha(\sigma, c, I)$  and  $\alpha(\text{delay}(\sigma, P_i, P_j, r), c, I)$ .*

Recall that  $S = \max\{3, \lceil \frac{n}{f} \rceil\}$  is the number of sets  $P_i$ . We define the following recursive function for every  $r$  and  $k$ ,  $1 \leq r \leq k$ :

$$m_{r,k} = \begin{cases} S & \text{if } r = k \\ (2(S-1) + 1)m_{r+1,k} + S & \text{if } 1 \leq r < k \end{cases}$$

A simple induction shows that  $m_{r,k} \leq (2S)^{k-r+1}$ .

The following lemma proves that  $m_{1,k}$  is the integer required in Lemma 2 for the message-passing model, by inductively constructing indistinguishability chains between executions where a set of processes may crash from a certain layer  $r$ .

**LEMMA 4.** *Let  $\sigma$  be a schedule with  $k$  layers such that for some  $r$ ,  $1 \leq r \leq k$ , no process is faulty in layers  $r, r+1, \dots, k$ . Then  $\alpha(\sigma, c, I) \approx_{m_{r,k}} \alpha(\text{crash}(\sigma, P_i, r), c, I)$  for every sequences of coins  $c$ , every initial configuration  $I$ , and every  $i \in \{1, \dots, S\}$ .*

**PROOF.** Let  $\sigma = \sigma_0$ . Throughout the proof we denote  $\alpha_i = \alpha(\sigma_i, c, I)$  for any schedule  $\sigma_i$ . The proof is by backwards induction on  $r$ .

*Base case:*  $r = k$ . We construct the following schedules. Let  $\sigma_1$  be the same as  $\sigma_0$  except that the messages sent by processes in  $P_i$  in the  $k$ -th layer are received by processes in  $P_{(i+1) \bmod S}$  only after the  $k$ -th layer, i.e.,  $\sigma_1 = \text{delay}(\sigma, P_i, P_{(i+1) \bmod S}, k)$ . By Lemma 3 we have  $\alpha_0 \stackrel{\mathcal{L}}{\approx} \alpha_1$ , for every process  $p \in P \setminus P_{(i+1) \bmod S}$ . We continue inductively to define schedules as above in the following way, for every  $h$ ,  $0 \leq h \leq S-1$ :  $\sigma_{h+1}$  is the same as  $\sigma_h$  except that the messages sent by processes in  $P_i$  in the  $k$ -th layer are received by processes in  $P_{(i+h+1) \bmod S}$  only after the  $k$ -th layer, i.e.,  $\sigma_{h+1} = \text{delay}(\sigma_h, P_i, P_{(i+h+1) \bmod S}, k)$ . By Lemma 3 we have  $\alpha_h \stackrel{\mathcal{L}}{\approx} \alpha_{h+1}$ , for every process  $p \in P \setminus P_{(i+h+1) \bmod S}$ .

Since in  $\sigma_S$  no messages sent by processes in  $P_i$  in layer  $k$  are ever received, then effectively processes in  $P_i$  are crashed in this layer:

$$\alpha_S = \alpha(\text{crash}(\sigma, P_i, k), c, I),$$

which implies that

$$\begin{aligned} \alpha(\sigma, c, I) &= \alpha_0 \approx_1 \alpha_1 \approx_1 \dots \approx_1 \alpha_S \\ &= \alpha(\text{crash}(\sigma, P_i, k), c, I). \end{aligned}$$

Therefore,  $\alpha(\sigma, c, I) \approx_S \alpha(\text{crash}(\sigma, P_i, k), c, I)$ .

*Induction step:* In general, we do the same as in the base case, except that we crash  $P_j$  in layer  $r+1$  before "erasing" messages from  $P_i$  to  $P_j$  in layer  $r$ , and afterwards revive  $P_j$  in layer  $r+1$ .

Formally, we assume that the lemma holds for layer  $r+1$ ,  $1 \leq r < k$ , and prove that it holds for layer  $r$ . Let  $\sigma_1 = \text{crash}(\sigma_0, P_{(i+1) \bmod S}, r+1)$ ; by the induction hypothesis,  $\alpha_0 \approx_{m_{r+1,k}} \alpha_1$ .

Let  $\sigma_2$  be the same as  $\sigma_1$  except that the messages received by processes in  $P_{(i+1) \bmod S}$  from processes in  $P_i$  in layer  $r$  are received only after the  $k$ -th layer, i.e.,  $\sigma_2 = \text{delay}(\sigma_1, P_i, P_{(i+1) \bmod S}, r)$ . By Lemma 3, at the end of layer  $r$  only processes in  $P_{(i+1) \bmod S}$  distinguish between the executions, but since they are crashed in layer  $r+1$  we have  $\alpha_1 \stackrel{\mathcal{L}}{\approx} \alpha_2$ , for every process  $p \in P \setminus P_{(i+1) \bmod S}$ , implying that  $\alpha_1 \approx_1 \alpha_2$ .

Let  $\sigma_3$  be the same as  $\sigma_2$ , except that the processes in  $P_{(i+1) \bmod S}$  do not crash in layer  $r+1$ . This implies that

$$\sigma_2 = \text{crash}(\sigma_3, P_{(i+1) \bmod S}, r+1).$$

By the induction hypothesis, we have  $\alpha_2 \approx_{m_{r+1,k}} \alpha_3$ .

We continue inductively to define schedules as above in the following way for every  $h$ ,  $0 \leq h \leq S-1$ . We define  $\sigma_{3h+1} = \text{crash}(\sigma_{3h}, P_{(i+h+1) \bmod S}, r+1)$ , and therefore by the induction hypothesis  $\alpha_{3h} \approx_{m_{r+1,k}} \alpha_{3h+1}$ . Let  $\sigma_{3h+2}$  be the same as  $\sigma_{3h+1}$  except that the messages received by processes in  $P_{(i+h+1) \bmod S}$  from processes in  $P_i$  in layer  $r$  are received only after the  $k$ -th layer, i.e.,  $\sigma_{3h+2} = \text{delay}(\sigma_{3h+1}, P_i, P_{(i+h+1) \bmod S}, r)$ . By Lemma 3, at the end of layer  $r$  only processes in  $P_{(i+h+1) \bmod S}$  distinguish between the executions, but since they are crashed in layer  $r+1$  we have  $\alpha_{3h+1} \stackrel{\mathcal{L}}{\approx} \alpha_{3h+2}$ , for every process  $p \in P \setminus P_{(i+h+1) \bmod S}$ , implying that  $\alpha_{3h+1} \approx_1 \alpha_{3h+2}$ .

Finally, we define  $\sigma_{3h+3}$  to be the same as  $\sigma_{3h+2}$ , except that processes in  $P_{(i+h+1) \bmod S}$  do not crash. This implies

that  $\sigma_{3h+2} = \text{crash}(\sigma_{3h+3}, P_{(i+h+1) \bmod S}, r+1)$ . By the induction hypothesis we have  $\alpha_{3h+2} \approx_{m_{r+1,k}} \alpha_{3h+3}$ .

The construction implies that in  $\sigma_{3(S-1)+2}$  the processes in  $P_i$  are effectively crashed from layer  $r$ , therefore

$$\alpha(\sigma_{3(S-1)+2}, c, I) = \alpha(\text{crash}(\sigma_0, P_i, r), c, I),$$

and hence

$$\begin{aligned} \alpha_0 &\approx_{m_{r+1,k}} \alpha_1 \approx_1 \alpha_2 \approx_{m_{r+1,k}} \alpha_3 \approx_{m_{r+1,k}} \dots \\ &\approx_{m_{r+1,k}} \alpha_{3(S-1)+1} \approx_1 \alpha_{3(S-1)+2}. \end{aligned}$$

Since  $m_{r,k} = (2(S-1)+1)m_{r+1,k} + S$ , this implies that  $\alpha_0 \approx_{m_{r,k}} \alpha(\text{crash}(\sigma_0, P_i, r), c, I)$ .  $\square$

At least  $n-f$  processes take a step in every layer, and hence every execution in the construction contains at least  $k(n-f)$  steps.

Lemmas 2 and 4 imply that for any sequence of coins  $C$ ,  $\alpha(\sigma_{full}, c, C_0) \approx_{S(2m_{1,k}+1)} \alpha(\sigma_{full}, c, C_S)$ . Since  $m_{1,k} \leq (2S)^k$ , substituting  $S(2m_{1,k}+1)$  in the parameter  $m$  of Theorem 1 yields that  $q_k \geq \frac{1}{(2S)^{k+1} + S + 1}$ .

Recall that  $S = \max\{3, \lceil \frac{n}{f} \rceil\}$ . Taking  $\lceil \frac{n}{f} \rceil$  to be a constant, we obtain the main result of this section:

**THEOREM 5.** *Let  $A$  be a randomized consensus algorithm in the asynchronous message passing model. There is a weak adversary and an initial configuration, such that the probability that  $A$  does not terminate after  $k(n-f)$  steps is at least  $\frac{1}{c^k}$ , where  $c$  is a constant if  $\lceil \frac{n}{f} \rceil$  is a constant.*

## 4. TRADEOFF FOR THE SHARED-MEMORY MODEL

We now derive a similar lower bound for two shared-memory models, where processes communicate through shared read/write registers. The first model consists of single-writer registers and a snapshot operation that costs one step, described formally in Subsection 4.1. In Subsection 4.2 we consider multi-writer registers.

Notice that in the shared-memory model, the definition of indistinguishability is slightly different than in the message-passing model – for two executions  $\alpha$  and  $\alpha'$  to be indistinguishable to a process  $p$ , we not only require  $p$  to have the same local states throughout both, but also that the values of the shared registers are the same throughout both (otherwise, for example, having  $p$  perform a read operation after  $\alpha$  and  $\alpha'$  might result in different executions). This implies that in both executions  $p$  performs the same shared-memory operations, including reading the same values from registers.

### 4.1 Single-Writer Cheap-Snapshot

We first consider a shared-memory model where processes communicate through single-writer registers. The lower bound is proved under a simplifying assumption that each read step accesses the registers of all processes. We call this the *single-writer cheap-snapshot* model, since each register is written to by one specific process, and all registers are read by any process in a single snapshot. This snapshot is charged one step, hence the term “cheap”.

As in a standard shared-memory model, a step of a process consists of accessing the shared memory, and performing local computations. We further assume that in the algorithm, the steps of every process alternate between a write and a cheap-snapshot, starting with a write. Any algorithm can

be transformed to satisfy this requirement by having a process rewrite the same value to its register if it is forced to take a write operation, or read all of the registers and ignore some of their values if it is forced to take a cheap-snapshot operation. This only doubles the step complexity.

Recall that the processes are partitioned into  $S = \max\{3, \lceil \frac{n}{f} \rceil\}$  sets  $P_1, \dots, P_S$ , each with at most  $f$  processes. We consider a restricted set of layered schedules.

**DEFINITION 3.** *A schedule  $\sigma$  is regular if for every layer  $L$  and every  $i$ ,  $1 \leq i \leq S$ , either all processes  $p \in P_i$  take a step in  $L$  consecutively, or none of the processes  $p \in P_i$  take a step in  $L$ . We denote by  $\pi$  the permutation of the sets  $P_i$  that take steps in  $L$ , i.e., if processes  $p \in P_i$  take a step in  $L$ , then  $\pi^{-1}(i)$  is their index in the layer. We denote by  $|\pi|$  the number of sets  $P_i$  that take steps in the layer.*

Regular schedules are useful in our proofs since in every layer, all the processes in some set  $P_i$  perform the same operation, as argued in the next lemma.

**LEMMA 6.** *Let  $\sigma$  be a regular schedule with  $k$  layers. Then in every layer  $L$  in  $\sigma$ , for every  $i$ ,  $1 \leq i \leq S$ , either all process  $p \in P_i$  do not take a step in  $L$ , or all processes  $p \in P_i$  perform a write operation in  $L$ , or all processes  $p \in P_i$  perform a cheap-snapshot operation in  $L$ .*

**PROOF.** The proof is by induction on the layer number  $r$ .

*Base case:* Let  $r = 1$ , i.e.,  $L$  is the first layer of  $\sigma$ . Since  $\sigma$  is regular, either all process  $p \in P_i$  take a step in  $L$ , or none of the processes  $p \in P_i$  take a step in  $L$ . If all take a step then by our assumption on the algorithm, it is a write operation. Otherwise, none take a step, which proves the base case.

*Induction step:* Assume the lemma holds for layer  $\ell$ ,  $1 \leq \ell \leq r$ . We prove the lemma for layer  $r+1$ . By the induction hypothesis, in every layer  $\ell$ ,  $1 \leq \ell \leq r$ , either all processes  $p \in P_i$  perform a cheap-snapshot operation, or all perform a write operation, or none perform an operation. If none perform any operation in any layer  $\ell \leq r$ , then at the beginning of layer  $r+1$  the pending operation of all processes  $p \in P_i$  is a write operation by our assumption on the algorithm. Otherwise, let  $\ell$  be the maximal layer in which all processes  $p \in P_i$  took a step. If they are cheap-snapshot operations then at the beginning of layer  $r+1$  the pending operation of all processes  $p \in P_i$  is a write operation by our assumption on the algorithm. If they are write operations then at the beginning of layer  $r+1$  the pending operation of all processes  $p \in P_i$  is a cheap-snapshot operation by our assumption on the algorithm. In any case, at the beginning of layer  $r+1$  either all processes  $p \in P_i$  have a pending cheap-snapshot operation, or all have a pending write operation. Since  $\sigma$  is regular, either none of the processes  $p \in P_i$  take a step in layer  $r+1$ , or all take a step in layer  $r+1$ , in which case it would either be a cheap-snapshot operation for all processes, or a write operation for all processes.  $\square$

In the proof, we apply certain manipulations to regular schedules, allowing us to delay and crash sets of processes, as follows.

**DEFINITION 4.** *Let  $\sigma$  be a schedule such that every  $p \in P_i$  is non-faulty in layer  $r$ , and such that  $P_i$  is not the last set of processes in the layer. Let  $\text{swap}(\sigma, P_i, r)$  be the schedule that is the same as  $\sigma$ , except that the steps of processes in  $P_i$  are*

swapped with steps of the next set of processes in that layer. Formally, if  $\pi$  is the permutation of layer  $r$  in  $\sigma$  and  $\pi'$  is the permutation of layer  $r$  in  $\text{swap}(\sigma, P_i, r)$ , and if  $j = \pi^{-1}(i)$ , then we have  $\pi'(j) = \pi(j+1)$  and  $\pi'(j+1) = \pi(j)$ .

Inductively, we define

$$\text{swap}^j(\sigma, P_i, r) = \text{swap}(\text{swap}^{j-1}(\sigma, P_i, r), P_i, r),$$

that is,  $P_i$  is swapped  $j$  times and moved  $j$  sets later in the layer.

**DEFINITION 5.** Let  $\sigma$  be a schedule. Let  $\text{delay}(\sigma, P_i, r)$  be the schedule that is the same as  $\sigma$ , except that the steps of  $P_i$  starting from layer  $r$  are delayed by one layer. Thus, there is no step of  $p \in P_i$  in layer  $r$ , the step of  $p \in P_i$  in layer  $r+1$  is the step that was in layer  $r$ , and so on. The permutations of the layers  $\ell \geq r+1$  do not change.

Delaying a set  $P_i$  from layer  $r$  can be seen as delaying  $P_i$  from layer  $r+1$ , swapping  $P_i$  in round  $r$  until it reaches the end of the layer, accounting for  $P_i$  as the first set in layer  $r+1$  instead of the last set in layer  $r$ , and then swapping  $P_i$  in round  $r+1$  until it reaches its original place in the layer.

Although accounting for  $P_i$  as the first set in layer  $r+1$  instead of the last set in layer  $r$  does not change the order of steps taken, it is technically a different schedule (recall that a schedule is defined as a sequence of layers, which in this case are different in layers  $r$  and  $r+1$ ). Therefore we define:

**DEFINITION 6.** Let  $\sigma$  be a schedule where the last set of processes in layer  $r$  is  $P_i$ , and this set does not appear in layer  $r+1$ . Let  $\text{rollover}(\sigma, P_i, r)$  be the schedule that is the same as  $\sigma$ , except that  $P_i$  is the first set in layer  $r+1$  instead of the last set in layer  $r$ .

Effectively, such two schedules  $\sigma$  and  $\text{rollover}(\sigma, P_i, r)$  have the same order of steps, which implies that the executions of these schedules is the same:

$$\alpha(\sigma, c, I) = \alpha(\text{rollover}(\sigma, P_i, r), c, I).$$

Definitions 4, 5, and 6 imply:

**COROLLARY 7.** Let  $\sigma$  be a regular schedule with  $k$  layers. For every  $r$ ,  $1 \leq r \leq k$ , and  $\pi_r$  the permutation of layer  $r$  in  $\sigma$ ,

$$\text{delay}(\sigma, P_i, r) = \text{swap}^{\pi_r^{-1}(i)-1}(\text{rollover}(\text{swap}^{|\pi_r|-\pi_r^{-1}(i)}(\text{delay}(\sigma, P_i, r+1), P_i, r), P_i, r), P_i, r+1).$$

Figure 1 shows an example of the schedules we go through when delaying a set  $P_i$  in layer  $r$  of a schedule  $\sigma$ .

Recall that  $\text{crash}(\sigma, P_i, r)$  is the schedule that is the same as  $\sigma$ , except that processes in  $P_i$  crash in layer  $r$ . Crashing a set  $P_i$  in layer  $r$  can be seen as delaying it from layer  $r$ , and then crashing it from layer  $r+1$ . Definitions 1 and 5 imply that:

**COROLLARY 8.** For every regular schedule  $\sigma$ ,

$$\text{crash}(\sigma, P_i, r) = \text{crash}(\text{delay}(\sigma, P_i, r), P_i, r+1).$$

An important property of regular schedules is that swapping, delaying, or crashing a set of processes  $P_i$  yields a regular schedule as well, because the sets are manipulated together.

**LEMMA 9.** Let  $\sigma$  be a regular schedule with  $k$  layers. Then for every  $i$ ,  $1 \leq i \leq S$ , and every  $r$ ,  $1 \leq r \leq k$ , the schedules  $\text{swap}(\sigma, P_i, r)$ ,  $\text{delay}(\sigma, P_i, r)$ ,  $\text{rollover}(\sigma, P_i, r)$ , and  $\text{crash}(\sigma, P_i, r)$  are regular.

**PROOF.** Every layer  $\ell \neq r$  in  $\text{swap}(\sigma, P_i, r)$  is the same as in  $\sigma$  and therefore satisfies the requirement of a regular schedule. In layer  $r$ , all processes that took steps in  $\sigma$  also take steps in  $\text{swap}(\sigma, P_i, r)$ , and each set remains consecutive. Therefore,  $\text{swap}(\sigma, P_i, r)$  is regular. It is also easy to see that  $\text{rollover}(\sigma, P_i, r)$  is regular.

The proof for  $\text{delay}(\sigma, P_i, r)$  and  $\text{crash}(\sigma, P_i, r)$  is by backwards induction on the layer number  $r$ .

*Base case:* For  $r = k$ , delaying a set  $P_i$  in the last layer, is the same as crashing  $P_i$ . Denote  $\sigma' = \text{delay}(\sigma, P_i, k) = \text{crash}(\sigma, P_i, k)$ . Every layer  $\ell < k$  in  $\sigma'$  is the same as in  $\sigma$ , and the last layer  $k$  is the same in  $\sigma'$  except that the processes in  $P_i$  do not take a step. Hence,  $\sigma'$  is also regular.

*Induction step:* We assume the lemma holds for every layer  $\ell$ ,  $r+1 \leq \ell \leq k$ , and prove it for layer  $r$ . By Corollary 7, the induction hypothesis and since swapping results in a regular schedule,  $\text{delay}(\sigma, P_i, r)$  is regular. By Corollary 8, the induction hypothesis and since delaying results in a regular schedule,  $\text{crash}(\sigma, P_i, r)$  is regular.  $\square$

We next construct an indistinguishability chain of schedules between any regular schedule and a schedule in which some set of processes is delayed or crashed. The construction relies on Corollary 7 and Corollary 8 to delay or crash a set of processes through a sequence of swaps. The elementary step in this construction, where a set is swapped with the following one, is provided by the next lemma.

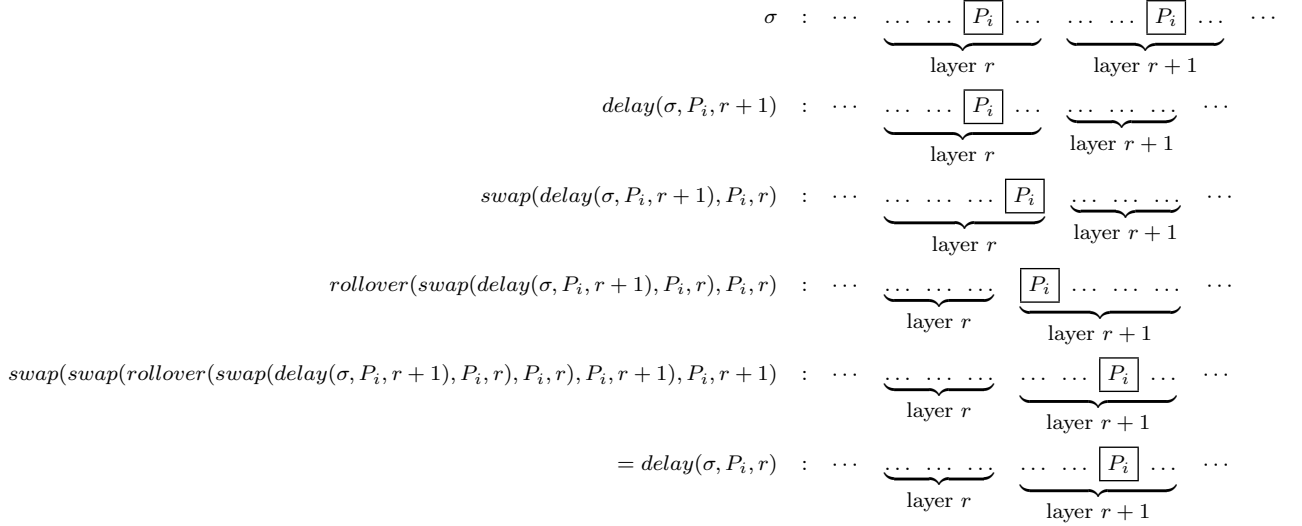
**LEMMA 10.** Let  $\sigma$  be a regular schedule with  $k$  layers. For any sequences of coins  $c$ , and initial configuration  $I$ , if  $P_i$  is not the last set of processes in layer  $r$ ,  $1 \leq r \leq k$ , then there is a set  $P_j$  such that at the end of layer  $r$  only processes in  $P_j$  distinguish between  $\alpha(\sigma, c, I)$  and  $\alpha(\text{swap}(\sigma, P_i, r), c, I)$ .

**PROOF.** Consider  $\text{swap}(\sigma, P_i, r)$  and let  $\pi$  be the permutation corresponding to layer  $r$ . Since  $P_i$  is not the last set in the layer, we have  $\pi^{-1}(i) \neq |\pi|$ . Let  $i' = \pi(\pi^{-1}(i) + 1)$ , i.e.,  $P_i$  is swapped with  $P_{i'}$ . By Lemma 6 either all the processes in  $P_i$  perform a cheap-snapshot operation or all processes in  $P_i$  perform a write operation. The same applies for  $P_{i'}$ .

If both types of operations are cheap-snapshot operations or both types are write operations (necessarily to different registers), then  $\alpha(\sigma, c, I) \stackrel{p}{\sim} \alpha(\text{swap}(\sigma, P_i, r), c, I)$  for every process  $p$  in the layer.

If one type of operations is cheap-snapshot and the other is writing, then only the processes in the set performing cheap-snapshot observe a difference. Denote this set by  $P_j$  (where  $j$  is either  $i$  or  $i'$ ).  $\square$

Notice that the set  $P_j$  (the value of the index  $j$ ) depends only on the types of operations performed, i.e, only on  $\sigma$ , and not on the sequences of coins  $c$  or the initial configuration  $I$ . This is necessary for claiming that the adversary is non-adaptive.



**Figure 1:** An example showing how *swap* operators are applied to delay a set of processes  $P_i$ ; assume  $P_i$  is the penultimate set in layer  $r$  and the third set in layer  $r+1$ . Note that the third transition does not modify the execution, and only accounts the steps of  $P_i$  to layer  $r+1$  instead of layer  $r$ ; the last transition just notes that we have obtained  $\text{delay}(\sigma, P_i, r)$ .

For every  $r$  and  $k$ ,  $1 \leq r \leq k$ , we define:

$$\begin{aligned}
s_{r,k} &= \begin{cases} 1 & \text{if } r = k \\ 2 \cdot c_{r+1,k} + 1 & \text{if } 1 \leq r < k \end{cases} \\
d_{r,k} &= \begin{cases} S & \text{if } r = k \\ d_{r+1,k} + S \cdot s_{r,k} + S \cdot s_{r+1,k} & \text{if } 1 \leq r < k \end{cases} \\
c_{r,k} &= \begin{cases} S & \text{if } r = k \\ d_{r,k} + c_{r+1,k} & \text{if } 1 \leq r < k \end{cases}
\end{aligned}$$

where  $S = \max\{3, \lceil \frac{n}{f} \rceil\}$  is the number of sets  $P_i$ . These recursive functions will be used for bounding the lengths of the indistinguishability chains in our construction.

The next proposition shows a bound on these functions, and a complete proof appears in the appendix.

PROPOSITION 11.  $c_{r,k} \leq (2S+4)^{k-r+1}$ .

The main technical result of this section is the following lemma, which will be used to show an indistinguishability chain between the executions that result from schedules  $\sigma$  and  $\text{crash}(\sigma, P_i, 1)$ , in order to apply Lemma 2. Additional claims, regarding *swap* and *delay*, are proved in order to carry through with the proof.

LEMMA 12. *Let  $\sigma$  be a regular schedule with  $k$  layers such that no process is faulty at any layer  $\ell \geq r$ , for some  $r$ ,  $1 \leq r \leq k$ . For any sequences of coins  $c$ , and initial configuration  $I$ , and for every  $i$ ,  $1 \leq i \leq S$ , the following all hold:*

$$\begin{aligned}
\alpha(\sigma, c, I) &\approx_{s_{r,k}} \alpha(\text{swap}(\sigma, P_i, r), c, I), \\
\alpha(\sigma, c, I) &\approx_{d_{r,k}} \alpha(\text{delay}(\sigma, P_i, r), c, I), \\
\alpha(\sigma, c, I) &\approx_{c_{r,k}} \alpha(\text{crash}(\sigma, P_i, r), c, I).
\end{aligned}$$

PROOF. Let  $\sigma_0 = \sigma$ . Throughout the proof, we denote  $\alpha_i = \alpha(\sigma_i, c, I)$  for every schedule  $\sigma_i$ , and  $\alpha'_i = \alpha(\sigma'_i, c, I)$  for every schedule  $\sigma'_i$ .

The proof is by backwards induction on  $r$ .

*Base case:*  $r = k$ . Consider  $\text{swap}(\sigma, P_i, k)$  where  $P_i$  is not the last set in the layer (otherwise swapping is undefined). By Lemma 10 there is a set  $P_j$ , which does not depend on  $c$  or  $I$ , such that  $\alpha(\sigma, c, I) \stackrel{p}{\approx} \alpha(\text{swap}(\sigma, P_i, r), c, I)$ , for every process  $p \notin P_j$ . Therefore,  $\alpha(\sigma, c, I) \approx_{s_{k,k}} \alpha(\text{swap}(\sigma, P_i, k), c, I)$ .

Delaying  $P_i$  in the last layer is equivalent to failing it, therefore  $\text{delay}(\sigma, P_i, k) = \text{crash}(\sigma, P_i, k)$ . Denote this schedule by  $\sigma'$ . We crash  $P_i$  by swapping it until it reaches the end of the layer and then removing it. In more detail, let  $\pi$  be the permutation of the last layer of  $\sigma$ , and define:

$$\sigma'' = \text{swap}^{|\pi| - \pi^{-1}(i)}(\sigma, P_i, k).$$

The proof of the base case for  $\text{swap}(\sigma, P_i, k)$  implies that there is a chain of length  $s_{k,k} \cdot (|\pi_r| - \pi_r^{-1}(i)) \leq (S-1) \cdot s_{k,k} = S-1$  between the executions, i.e.,  $\alpha_0 \approx_{S-1} \alpha(\sigma'', c, I)$ . Clearly,  $\alpha(\sigma'', c, I) \stackrel{p}{\approx} \alpha(\sigma', c, I)$ , for every process  $p \notin P_i$ , and therefore,  $\alpha(\sigma, c, I) \approx_{d_{k,k}} \alpha(\text{delay}(\sigma, P_i, r), c, I)$  and  $\alpha(\sigma, c, I) \approx_{c_{k,k}} \alpha(\text{crash}(\sigma, P_i, r), c, I)$ .

*Induction step:* Assume the lemma holds for layer  $r+1 \leq k$ . We prove that it holds for layer  $r$ .

We first deal with swapping; assume that  $P_i$  is not the last set in the layer and consider  $\text{swap}(\sigma, P_i, r)$ . By Lemma 10, there is a set  $P_j$ , which does not depend on  $c$  or  $I$ , such that at the end of layer  $r$  only process in  $P_j$  distinguish between  $\alpha(\sigma, c, I)$  and  $\alpha(\text{swap}(\sigma, P_i, r), c, I)$ . We define  $\sigma_1$  to be the same as  $\sigma$  except that processes in  $P_j$  are crashed in layer  $r+1$ , i.e.,  $\sigma_1 = \text{crash}(\sigma, P_j, r+1)$ . By the induction hypothesis,  $\alpha_0 \approx_{c_{r+1,k}} \alpha_1$ . Let  $\sigma_2$  be the same as  $\sigma_1$  except that  $P_i$  and  $P_j$  are swapped in layer  $r$ , i.e.,  $\sigma_2 = \text{swap}(\sigma_1, P_i, r)$ . Since only processes in  $P_j$  observe the swapping, but are all crashed in the next layer, we have that  $\alpha_1 \stackrel{p}{\approx} \alpha_2$  for every process  $p \notin P_j$ . Finally, let  $\sigma_3$  be the same as  $\sigma_2$ , except that processes in  $P_j$  are not crashed in layer  $r+1$ , i.e.,  $\sigma_2 =$

$crash(\sigma_3, P_j, r+1)$ . By the induction hypothesis,  $\alpha_2 \approx_{c_{r+1,k}} \alpha_3$ . Notice that  $\sigma_3 = swap(\sigma, P_i, r)$ , and  $2c_{r+1,k} + 1 = s_{r,k}$ , which implies that  $\alpha(\sigma, c, I) \approx_{s_{r,k}} \alpha(swap(\sigma, P_i, r), c, I)$ .

Next, we consider the case of delaying a process, i.e.,  $delay(\sigma, P_i, r)$ . (Recall Figure 1.) By Corollary 7,

$$delay(\sigma, P_i, r) = swap^{\pi_{r+1}^{-1}(i)-1}(rollover^{|\pi_r|-\pi_r^{-1}(i)}(delay(\sigma, P_i, r+1), P_i, r), P_i, r), P_i, r+1).$$

Recall that applying *rollover* does not change the execution. Hence, by the proof for swapping, the induction hypothesis, and since

$$\begin{aligned} d_{r+1,k} + s_{r,k} \cdot (|\pi_r| - \pi_r^{-1}(i)) + s_{r+1,k} \cdot (\pi_{r+1}^{-1}(i) - 1) \\ \leq d_{r+1,k} + S \cdot s_{r,k} + S \cdot s_{r+1,k} = d_{r,k} \end{aligned}$$

it follows that  $\alpha(\sigma, c, I) \approx_{d_{r,k}} \alpha(delay(\sigma, P_i, r), c, I)$ .

Finally, we consider the case of crashing a process, i.e.,  $crash(\sigma, P_i, r)$ . By Corollary 8,

$$crash(\sigma, P_i, r) = crash(delay(\sigma, P_i, r), P_i, r+1).$$

By the proof for delaying, the induction hypothesis, and since  $d_{r,k} + c_{r+1,k} = c_{r,k}$ , it follows that

$$\alpha(\sigma, c, I) \approx_{c_{r,k}} \alpha(crash(\sigma, P_i, r), c, I).$$

□

In every layer, at most  $n/S \leq f$  processes do not take a step. That is, at least  $n - f$  processes take a step in every layer, and hence, every execution in the construction contains at least  $k(n - f)$  steps.

Lemmas 2 and 12 imply that for every sequence of coins  $c$ ,  $\alpha(\sigma_{full}, c, C_0) \approx_{S(2c_{1,k}+1)+1} \alpha(\sigma_{full}, c, C_S)$ . Since  $c_{1,k} \leq (2S+4)^k$ , we have that  $S(2c_{1,k}+1)+1 \leq (2S+4)^{k+1}$ . Substituting in Theorem 1 yields that  $q_k \geq \frac{1}{(2S+4)^{k+1}}$ . Since  $S$  can be taken to be a constant when  $\lceil \frac{n}{f} \rceil$  is a constant, we get the next theorem:

**THEOREM 13.** *Let  $A$  be a randomized consensus algorithm in the asynchronous shared-memory model, with single-writer registers and cheap snapshots. There is a weak adversary and an initial configuration, such that the probability that  $A$  does not terminate after  $k(n - f)$  steps is at least  $\frac{1}{c^k}$ , where  $c$  is a constant if  $\lceil \frac{n}{f} \rceil$  is a constant.*

## 4.2 Multi-Writer Cheap-Snapshot

We derive the lower bound for multi-writer registers by reduction to single-writer registers. We use Algorithm 28 from [5, Section 10.2.3] to simulate multi-writer registers from single-writer registers. In this algorithm, performing a high-level read or write operation (to a multi-writer register) involves  $n$  low-level read operations (of all single-writer registers) and possibly one low-level write operation (to the process' own single-writer register). Thus, the original simulation multiplies the total step complexity by  $O(n)$ .

However, with cheap-snapshots, we can read all single-writer registers in one step, yielding a simulation that only doubles the total step complexity (since writing includes a cheap-snapshot operation). Combining this with Theorem 13 yields the following theorem.

**THEOREM 14.** *Let  $A$  be a randomized consensus algorithm in the asynchronous shared-memory model, with single-writer registers and cheap snapshots. There is a weak*

*adversary and an initial configuration, such that the probability that  $A$  does not terminate after  $k(n - f)$  steps is at least  $\frac{1}{c^k}$ , where  $c$  is a constant if  $\lceil \frac{n}{f} \rceil$  is a constant.*

## 5. MONTE-CARLO ALGORITHMS

Another way to overcome the impossibility of asynchronous consensus, is to allow Monte-Carlo algorithms. This requires us to relax the agreement property and allow the algorithm to decide on conflicting values, with small probability. Let  $\epsilon_k$  be the probability that processes decide on conflicting values after  $k(n - f)$  steps. The following is a theorem similar to Theorem 1, for bounding the probability of terminating after  $k(n - f)$  steps.

**THEOREM 15.** *Assume there is an integer  $m$  such that for any sequences of coins  $c$ ,  $\alpha(\sigma_{full}, c, C_0) \approx_m \alpha(\sigma_{full}, c, C_S)$ . Then  $q_k \geq \frac{1-\epsilon_k}{m+1}$ .*

**PROOF.** Assume, by way of contradiction, that  $(m+1)q_k < 1 - \epsilon_k$ . Consider the  $m+1$  executions in the sequence implied by the fact that  $\alpha(\sigma_{full}, c, C_0) \approx_m \alpha(\sigma_{full}, c, C_S)$ . The probability that  $A$  does not terminate in at least one of these  $m+1$  executions is at most  $(m+1)q_k$ . By assumption,  $q_k(m+1) < 1 - \epsilon_k$ , and hence, the set  $B$  of sequences of coins  $c$  such that  $A$  terminates in all  $m+1$  executions has probability  $\Pr[c \in B] > \epsilon_k$ .

If the agreement property is satisfied in all  $m+1$  executions, then by the validity condition, as in the proof of Theorem 1, we get that the decision in  $\alpha(\sigma_{full}, c, C_0)$  is the same as the decision in  $\alpha(\sigma_{full}, c, C_S)$ , which is a contradiction. Hence, for every  $c \in B$ , the agreement condition does not hold in at least one of these executions. But this means that  $A$  satisfies agreement with probability smaller than  $1 - \epsilon_k$ , which is a contradiction. □

For example, the algorithms for the message-passing model given by Kapron et al. [22] are Monte-Carlo, i.e., have a small probability for terminating without agreement. The bound we obtain in Theorem 15 on the probability  $q_k$  of not terminating increases as the allowed probability  $\epsilon_k$  of terminating without agreement decreases, and coincides with Theorem 1 in case the agreement property must always be satisfied.

## 6. DISCUSSION

We presented lower bounds for the termination probability achievable by randomized consensus algorithms with bounded running time, under a very weak adversary. Our results are particularly relevant in light of the recent surge of interest in providing *Byzantine fault-tolerance* in practical, asynchronous distributed systems (e.g., [11, 25]). The adversarial behavior in these applications is better captured by non-adaptive adversaries as used in our lower bounds, rather than the adaptive adversary, which can observe the results of local coin-flips, used in most previous lower bounds [1, 4, 9].

For all models we have shown that the probability  $q_k$  that the algorithm fails to complete in  $k(n - f)$  steps is at least  $\frac{1}{c^k}$ , for a model-dependent value  $c$  which is a constant if  $\lceil \frac{n}{f} \rceil$  is a constant. Table 1 shows the bounds for specific values of  $k$ .

The previous lower bound for the synchronous message-passing model [16] is  $q_k \geq \frac{1}{(ck)^k}$ , for some constant  $c$ . From the perspective of the expected total step complexity, given

	Model	$k = \log \log n$	$k = \log n$	$k = \log^2 n$
lower bound	asynchronous MP, SWCS, MWCS	$\frac{1}{\log^{\Omega(1)} n}$	$\frac{1}{n^{\Omega(1)}}$	$\frac{1}{n^{\Omega(\log n)}}$
	synchronous MP [16]	$\frac{1}{(\log n)^{\Omega(\log \log \log n)}}$	$\frac{1}{n^{\Omega(\log \log n)}}$	
upper bound	MP [22]	$\frac{1}{\log^{O(1)} n}$	$\frac{1}{n^{O(1)}}$	
	SWMR [7]			$\frac{1}{n^{O(1)}}$
	MWMR [6]		$\frac{1}{n^{O(1)}}$	

**Table 1: Bounds on  $q_k$  in different models.** MP is the message-passing model, while SW/MW stands for single/multi-writer registers, SR/MR for single/multi-reader registers, and CS for cheap snapshots.

a non-termination probability  $\delta$ , the lower bound of [16] implies  $\Omega\left((n-f)\frac{\log 1/\delta}{\log \log 1/\delta}\right)$  steps, which is improved by our bound to  $\Omega((n-f)\log 1/\delta)$  steps.

For the asynchronous message passing model, Kapron et al. [22] show algorithms with probability  $1 - \frac{1}{\text{poly}(n)}$  for terminating in  $2^{\Theta(\log^7 n)}$  (asynchronous) rounds, and probability  $1 - \frac{1}{\text{poly}(\log(n))}$  for terminating in polylogarithmic number of (asynchronous) rounds.

In the shared-memory model with single-writer multi-reader registers, Aumann and Bender [7] show a consensus algorithm with probability  $1 - \frac{1}{n^{O(1)}}$  for terminating in  $O(n \log^2 n)$  steps. For multi-writer multi-reader registers, Aumann [6] presents an iterative consensus algorithm, with constant probability to terminate at each iteration, independently of the previous iterations. This implies that the probability of terminating after  $k$  iterations is  $1 - \frac{1}{c^k}$ , for some constant  $c$ .

Tightening the bounds, especially for the synchronous model and for large values of  $k$ , is the main technical open question we leave for further research. Our lower bounds can be used to estimate the error distribution and bound the variance of the running time of randomized consensus algorithms. They do not yield significant lower bounds for the expected step complexity—there is still a large gap between the (trivial) lower bounds and upper bounds for the shared-memory model, with a weak adversary.

Another, broader research direction is to explore complexity bounds on randomized algorithms for other coordination problems, most notably *renaming* [3] and *set consensus* [13].

**Acknowledgements.** We would like to thank Valerie King for drawing our attention to [16] and for interesting discussions, James Aspnes for coining the term *cheap-snapshot*, and David Hay and Rotem Oshman for helpful comments.

## 7. REFERENCES

- [1] J. Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *J. ACM*, 45(3):415–450, May 1998.
- [2] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–176, Sept. 2003.
- [3] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *J. ACM*, 37(3):524–548, July 1990.
- [4] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. In *Proceedings of the 39th annual ACM symposium on Theory of computing (STOC)*, pages 155–164, 2007.
- [5] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. McGraw-Hill, 1st edition, 1998.
- [6] Y. Aumann. Efficient asynchronous consensus with the weak adversary scheduler. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 209–218, 1997.
- [7] Y. Aumann and M. A. Bender. Efficient low-contention asynchronous consensus with the value-oblivious adversary scheduler. *Distributed Computing*, 17(3):191–207, Mar. 2005.
- [8] Y. Aumann and A. Kapah-Levy. Cooperative sharing and asynchronous consensus using single-reader single-writer registers. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 61–70, 1999.
- [9] Z. Bar-Joseph and M. Ben-Or. A tight lower bound for randomized synchronous consensus. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 193–199, 1998.
- [10] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine agreement in the full-information model in  $o(\log n)$  rounds. In *Proceedings of the 38th annual ACM symposium on Theory of computing (STOC)*, pages 179–186, 2006.
- [11] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
- [12] T. D. Chandra. Polylog randomized wait-free consensus. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 166–175, 1996.
- [13] S. Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.
- [14] L. Cheung. Randomized wait-free consensus using an atomicity assumption. In *Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS)*, pages 47–60, 2005.
- [15] B. Chor, A. Israeli, and M. Li. On processor coordination using asynchronous hardware. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 86–97, 1987.
- [16] B. Chor, M. Merritt, and D. B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *J. ACM*, 36(3):591–614, July 1989.
- [17] D. Dolev and H. R. Strong. Authenticated algorithms

- for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
- [18] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.
- [19] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.
- [20] S. Goldwasser, E. Pavlov, and V. Vaikuntanathan. Fault-tolerant distributed computing in full-information networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 15–26, 2006.
- [21] M. Herlihy. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst.*, 13(1):124–149, January 1991.
- [22] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1038–1047, 2008.
- [23] A. Karlin and A. C.-C. Yao. Probabilistic lower bounds for byzantine agreement and clock synchronization. Unpublished manuscript.
- [24] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 990–999, 2006.
- [25] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 45–58, 2007.
- [26] M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, pages 163–183, 1987.
- [27] Y. Moses and S. Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002.

## APPENDIX

Recall the definition of the following recursive functions of  $r$  and  $k$ , for  $1 \leq r \leq k$ .

$$\begin{aligned}
 s_{r,k} &= \begin{cases} 1 & \text{if } r = k \\ 2 \cdot c_{r+1,k} + 1 & \text{if } 1 \leq r < k \end{cases} \\
 d_{r,k} &= \begin{cases} S & \text{if } r = k \\ d_{r+1,k} + S \cdot s_{r,k} + S \cdot s_{r+1,k} & \text{if } 1 \leq r < k \end{cases} \\
 c_{r,k} &= \begin{cases} S & \text{if } r = k \\ d_{r,k} + c_{r+1,k} & \text{if } 1 \leq r < k \end{cases}
 \end{aligned}$$

We bound these functions as follows.

**Proposition 11**  $c_{r,k} \leq (2S + 4)^{k-r+1}$ .

PROOF. By definition of  $s_{r,k}$  we have  $s_{r,k} = 2c_{r+1,k} + 1$  which implies that

$$\begin{aligned}
 d_{r,k} &= d_{r+1,k} + S \cdot s_{r,k} + S \cdot s_{r+1,k} \\
 &= d_{r+1,k} + S \cdot (2c_{r+1,k} + 1) + S \cdot (2c_{r+2,k} + 1) \\
 &= d_{r+1,k} + 2Sc_{r+1,k} + 2Sc_{r+2,k} + 2S.
 \end{aligned}$$

Substituting this in the definition of  $c_{r,k}$  gives

$$\begin{aligned}
 c_{r,k} &= c_{r+1,k} + d_{r,k} \\
 &= c_{r+1,k} + d_{r+1,k} + 2Sc_{r+1,k} + 2Sc_{r+2,k} + 2S.
 \end{aligned}$$

Since  $d_{r+1,k} = c_{r+1,k} - c_{r+2,k}$ , we have

$$\begin{aligned}
 c_{r,k} &= c_{r+1,k} + d_{r+1,k} + 2Sc_{r+1,k} + 2Sc_{r+2,k} + 2S \\
 &= c_{r+1,k} + (c_{r+1,k} - c_{r+2,k}) \\
 &\quad + 2Sc_{r+1,k} + 2Sc_{r+2,k} + 2S \\
 &= (2S + 2)c_{r+1,k} + (2S - 1)c_{r+2,k} + 2S,
 \end{aligned}$$

where the initial values are  $c_{k,k} = S$ , and

$$\begin{aligned}
 c_{k-1,k} &= c_{k,k} + d_{k-1,k} \\
 &= c_{k,k} + (d_{k,k} + S \cdot s_{k-1,k} + S \cdot s_{k,k}) \\
 &= c_{k,k} + (d_{k,k} + S \cdot (2c_{k,k} + 1) + S \cdot s_{k,k}) \\
 &= S + S + S \cdot (2S + 1) + S \cdot 1 = S(2S + 4).
 \end{aligned}$$

We now prove the claim by backwards induction on  $r$ .

*Base case:* For  $r = k$  we have  $c_{k,k} = S \leq (2S + 4)$ , and for  $r = k - 1$  we have  $c_{k-1,k} = S(2S + 4) \leq (2S + 4)^2$ .

*Induction step:* We assume the lemma holds for every  $\ell$  such that  $r + 1 \leq \ell \leq k$ , and prove it for  $r$ .

Substituting the induction hypothesis on  $c_{r+1,k}$  and  $c_{r+2,k}$  gives:

$$\begin{aligned}
 c_{r,k} &= (2S + 2)c_{r+1,k} + (2S - 1)c_{r+2,k} + 2S \\
 &\leq (2S + 2)(2S + 4)^{k-r} \\
 &\quad + (2S - 1)(2S + 4)^{k-r-1} + 2S \\
 &\leq (2S + 2)(2S + 4)^{k-r} + (2S + 4)^{k-r} + (2S + 4)^{k-r} \\
 &= (2S + 2 + 1 + 1)(2S + 4)^{k-r} \\
 &= (2S + 4)^{k-r+1},
 \end{aligned}$$

which completes the proof.  $\square$