From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement*

Eli Gafni[1] Rachid Guerraoui[2] Bastian Pochon[2]
eli@cs.ucla.edu rachid.guerraoui@epfl.ch bastian.pochon@epfl.ch
(1) Department of Computer Science, UCLA
(2) Distributed Programming Laboratory, EPFL

Abstract

Set agreement, where processors decisions constitute a set of outputs, is notoriously harder to analyze than consensus where the decisions are restricted to a single output. This is because the topological questions that underly set agreement are not about simple connectivity as in consensus. Analyzing set agreement inspired the discovery of the relation between topology and distributed algorithms, and consequently the impossibility of set agreement.

Yet, the application of topological reasoning has been to the static case, that of asynchronous and synchronous tasks. It is not known yet for example, how to characterize starvation-free solvability of non-terminating tasks. Non-terminating tasks are dynamic entities with no defined end. In a similar vain, early deciding synchronous set agreement, in which the number of rounds it takes a processor to decide adapts to the actual number of failures, falls in this category of dynamic entities.

This paper develops a simulation technique that brings to bear topological results to deal with the dynamic situation that arises with early decisions. The novelty of the new simulation is the ability of simulators to look back at the transcript of past rounds of the simulation to influence their current behavior. Using our new technique, we not only re-derive past results, but we propose and prove a lower bound to synchronous early stopping set agreement. We then provide an algorithm to match the lower bound. Our technique uses the BG simulation, in the most creative way it was used to-date, to obtain a rather simple reduction from a static asynchronous impossibility. This reduction is a simple alternative to yet unknown topological argument, and in fact may suggest the way of finding such an argument.

*Technical Report ID: IC/2004/93

1 Introduction

Results about the complexity of set agreement are intriguing, as they present an intrinsic trade-off between the number of processors in a system, the degree of coordination that these processors can reach, and the number of failures that are tolerated [7]. The complexity of early deciding [9] set agreement is even more intriguing as it brings to the picture the number of failures that actually occur in a given computation.

Set agreement is a generalization of the widely studied consensus problem [11]. In set agreement, each processor is supposed to propose a value, and eventually decide on some output that was initially proposed, such that every correct processor eventually decides (just like in consensus). Processors are restricted not to decide on more than k distinct outputs. We talk about k-set agreement, and consensus is the special case where $k = 1$.

Set agreement was introduced in [6] where it was conjectured that, in an asynchronous model,[1] the problem has a solution if and only if strictly less than k processors may crash. This conjecture has sparked a fruitful line of research, applying algebraic topology arguments to distributed computing [3, 14, 17]. After proving the conjecture, researchers applied topological arguments to prove a lower bound on the complexity of set agreement in the synchronous model [7, 12, 13].[2] In short, the result states that any synchronous k-set agreement algorithm that tolerates t failures (where $t < N$ and N is the total number of processors in the system) has at least one run where at least one correct processor does not reach a decision before round $\lfloor t/k \rfloor + 1$. This lower bound does not however say much about the existence of algorithms that would expedite a decision in runs where f ($f < t$) failures actually occur. In particular, one would expect that, in runs where few failures occur, a decision can be reached earlier than in those with more failures. Roughly speaking, early deciding algorithms are those that have that adaptive flavor: their efficiency depends on the effective number of failures that occur in a given computation, rather than (only) on the (total) number of failures that can be tolerated [9]. In practice, failures rarely happen, and it makes sense to devise algorithms that decide earlier when fewer failures occur. For consensus, a significant efficiency improvement has been established when considering the effective number of failures [5, 10, 15]. In particular, it was shown that for any integer $f \leq t$, any consensus algorithm has a run with at most f failures in which some processor decides not earlier than in round $min(f + 2, t + 1)$, and there is an algorithm where all correct processors decide by round $min(f + 2, t + 1)$ for all f [5, 15].

Here we consider set agreement, yet even for consensus, the early deciding synchronous lower bound is involved. The bound has been argued recently [15] in an ad-hoc manner through similarity between computations, rather than by using the more modern methods developed with the emergence of the topological techniques [3, 4, 7, 14].

This is not surprising. Early decision argumentation is evocative of the analysis that is called for when arguing one-shot vs. long-lived object implementations [2]. The issue there is whether a processor can obtain an output in face of continual arrival and departure of other processors. This is in fact not surprising as the topological method as to-date has not helped in resolving this matter. The topological method characterizes the topological structure of views of processors at the end of the computation, and has not been adapted yet to deal with evolving computation. The early decision question seems to fall into the category of evolving dynamic computations. Hence, the lack of lower bound for early deciding set agreement, and the involvement of the early deciding synchronous consensus lower bound.

[1] In an asynchronous model of distributed computation, (1) processors execute the algorithm assigned to them unless they crash, in which case they stop all their activities and they are said to be faulty (not correct), as well as (2) there is no bound on processor relative speed and message communication delay.

[2] In a synchronous model of distributed computation, processors execute in a lockstep manner, moving incrementally from a round to the next, and exchanging messages in every round; if a processor p does not receive a message from a processor q in a round r, then no processor ever receives any message from q in any subsequent round $r' > r$.

This paper proposes to deal with the dynamic situation of early decision in a round-about way, rather than through a head-on attack. It does not apply topology directly. Rather, similar to [12] which uses algorithmic reduction, it reasons about the dynamics of the synchronous computation through reduction. Unlike [12] where the simulation proceeds forward without "looking back", here we propose a simulation technique using the BG-agreement protocol [3, 4]. This allows simulators to go back and look at the transcript of the simulation (in shared memory) and by that allows us to argue about the *dynamics* of the computation rather than just its *end*.

Our result supports the tradition in computer-science that once few cornerstone impossibility or complexity results have been proved using direct (topological) arguments, from there on one should use reductions rather than argue (topology) anew. In the same vain that one proves NP-completeness by reduction rather than rehashing Cook's proof of the SAT NP-completeness [8].

Nevertheless, it is intriguing to understand the analogue of our simulation in the pure topological domain. Unfortunately, our experience shows that the distributed domain is loaded with semantics and interpretations and that consequently finding the topological analogue may not be an easy undertaking. On the encouraging side, we hold the hope that the technique we present in this paper will prove useful in arguing about non-terminating tasks.

The rest of the paper is organized as follows. Section 2 gives some preliminaries about distributed computing models that are needed to state and prove our results. Section 3 states and proves our lower bound result. Section 4 presents our optimal early deciding algorithm. Due to space limitation, the correctness proof of the algorithm is postponed to the optional Appendix.

2 Preliminaries

In the following we present the main elements of the synchronous message-passing model, in which we state the lower bound and design our optimal algorithm, and the asynchronous shared memory model, which we use for the reduction in our lower bound proof. Then we briefly recall the set agreement problem, and finally we present the BG-agreement protocol, the simulation technique that underlies our lower bound proof.

2.1 The Synchronous Message-passing Model

We consider a set of $N = n + 1$ processors $\Pi = \{p_0, \ldots, p_n\}$. Processors communicate by message-passing. Communication channels are reliable. Processors execute in a synchronous, round-based model [16]. A run is a sequence of rounds. Every round is composed of three phases. In the first phase, every processor broadcasts a message to all the other processors. In the second phase, every processor receives all the messages sent to it during the round. In the third phase, every processor may perform a local computation, before starting the next round. Processors may fail by crashing. A processor that crashes does not execute any step thereafter, and is said to be *faulty*. Processors that do not crash are said to be *correct*. When processor p_i crashes in round r, a subset of the messages that p_i sends in round r (possibly the empty set) is received by the end of round r. A message broadcast in round r by a processor that does not crash in round r is received, at the end of round r, by every processor that reaches the end of round r. We say that a processor p_i *sees* f crashes at the end of any round r, if p_i receives messages from all processors but f. We consider that there are at most $t < N$ processors that may fail in any run. The *state* of a processor p_i, at the end of round r, consists in the content of its local memory, including the messages received in each round $r' \leq r$, as well as the local variables pf p_i.

2.2 The Asynchronous Shared Memory Model

We prove our result by reducing computations in the synchronous message-passing model, recalled above, to computations in the asynchronous shared memory model, which we recall here. For clarity reasons, processors are called *simulators* in the asynchronous shared memory model. Precisely, we consider a set of $k+1$ simulators $\{sim_0, \ldots, sim_k\}$. Simulators communicate through asynchronous shared memory. In the asynchronous shared memory model, there exists no bound on the processor relative execution speed. Shared memory is organized in cells (sometimes called registers), where each memory cell may contain an infinite number of bits. Cells of the shared memory support three operations: the write(v) operation atomically writes value v into the cell; the read() operation atomically returns the content of the cell; the snapshot() operation returns an atomic view of all the cells (i.e., at a certain point in time between the invocation to the operation and the return of the operation) [1]. Any cell may be written by a single simulator, and read by all of them. To simplify the presentation, we assume in the following that after executing an operation snapshot(*args*), the variables *args* are accessible by the simulator in its local memory with the content as by the time of the snapshot() operation. Without loss of generality, we consider that the simulators execute full-information protocols in shared memory [14]. In a full-information protocol, any simulator sim_i writes its entire state into a memory cell, whenever sim_i writes anything into this cell. Any simulator that later reads the cell reads the entire history of the states of simulator sim_i.

2.3 The k-Set Agreement Problem

We recall here the k-set agreement problem. Each processor proposes a value v from a set of inputs V, and is supposed to eventually decide on an output v' of V, such that every output is a proposed value, and there are at most k distinct outputs. Solving k-set agreement in a wait-free manner means that every correct processor eventually decides (no matter how many processors fail). Wait-free k-set agreement is proved impossible in an asynchronous shared memory model of $k+1$ processors [3, 14, 17].

2.4 The BG-Agreement Protocol

In our lower bound proof that follows, we make extensive use of the BG-agreement notion [3, 4]. For completeness and self-containment of our lower bound proof we briefly review this notion here.

A BG-agreement protocol is a distributed algorithm in the asynchronous shared memory model. Basically the protocol consists of a wait-free code, with the exception of the last statement of the code, which is a wait statement. The BG-agreement protocol is an election protocol. It elects one of the participating processors, which is called the winner. Consequently, if each participating processor writes a proposal in shared memory before starting the protocol, the protocol decides on one of the proposals. The protocol is guaranteed to elect a leader when all participating processors arrive at the wait statement. While waiting for other processors to reach the wait statement, the outcome of the election may not be known and, in our terminology, the BG-agreement is not *resolved*. Consequently, if the BG-agreement is not resolved, one of the participating processors must be in the middle of the code rather than at the wait statement (we say that this processor is *blocking* the BG-agreement). Thus, if processors that are waiting time-share and execute other protocols while waiting, and the BG-agreement is not resolved, we can conclude that at least one processor does not participate in other protocols.

An instantiation of the BG-agreement protocol is illustrated in Figure 1. Variables v_i, x_i and S_i (for any $0 \leq i \leq n$) are in shared memory, written by simulator sim_i and read by all. The $*$ in front of the parameter *result* indicates an output parameter. The wait statement spans over lines 7 to 9. A processor proposes a value v to a BG-agreement instance by invoking BGpropose(v, *result*) and expects the result of the agreement to be stored in local variable *result*. The intuitive idea underlying how the BG-agreement protocol goes is as follows: a processor writes its proposed value and its identifier in

```
1: in shared memory: v_i ∈ V, init ⊥, x_i ∈ {true, false}, init false, S_i ⊆ {0, ..., n}, init ∅

2: procedure BGpropose(v, *result)
3:    v_i := v
4:    x_i := true
5:    snapshot(x_1, ..., x_n)
6:    S_i := { j | x_j = true, 0 ≤ j ≤ n }
7:    do                                   {The do loop is the wait statement}
8:      snapshot(S_0, ..., S_n)
9:    until ∀j ∈ S_i : S_j ≠ ∅
10:   winner := min(S_j), where j ∈ S_i and ∀k ∈ S_i : |S_k| ≥ |S_j|
11:   *result := v_winner
```

Figure 1: BG-agreement protocol (code for simulator sim_i)

BG-agreement in $R_{r,1}$	
Purpose	agree upon the state of a processor p_j at the beginning of round $r+1$ (i.e., whether p_j crashes in round r and, if not, which messages p_j receives in round r)
Input values	"failed", "p_j receives messages from all processors in a set $correct \subseteq 2^{\Pi}$"

BG-agreement in $R_{r,2}$	
Purpose	agree upon a correct processor at the beginning of round $r+1$
Input values	"no processor", "kill $p_l \in \Pi$"

Figure 2: Series of BG-agreements in $R_{r,1}$ and $R_{r,2}$

shared memory (we say that the processor "registers"), takes a snapshot of the registered processors, and writes its snapshot into shared memory. The processor then continuously takes snapshots of the shared memory, until all the registered processors have written their snapshots into shared memory. The processor then returns the value of the processor with the smallest identifier in the smallest set corresponding to the snapshot of a processor.

3 The Lower Bound

Theorem 1 *For any integer $f \geq 0$ there is no algorithm $C(k, f)$ that solves k-set agreement in a synchronous message-passing model, under the following conditions:*

1. *In runs in which eventually no more than $k-1$ processors fail in each round, eventually every correct processor decides.*
2. *A processor that sees f failures for some fixed f, decides at the latest after $\lfloor f/k \rfloor + 1$ rounds.*

Theorem 1 generalizes the result of [5, 15] on early deciding consensus. Indeed taking $k = 1$ in the above theorem leads to the early deciding lower bound of consensus.

The proof is by contradiction and the main idea is to reduce the problem of solving wait-free k-set agreement in the asynchronous shared memory model to an algorithm $C(k, f)$ solving k-set agreement and satisfying the two conditions of Theorem 1. In short the reduction consists in simulating, with

```
1: In shared memory: state_{r,j}, init ⊥, FinalFaulty_{r,j}, init ∅, r ≥ 1, 0 ≤ j ≤ n

2: procedure Simulate(C, f)
3:    r := 0, Correct := Π
4:    {                                          {Execute two coroutines in parallel}
5:    ResolveInputs()                            {Coroutine 1: the simulation}
6:    for r := 1 to ∞ do
7:       r := +1
8:       Execute R_{r,1}
9:       Execute R_{r,2}
10:      SimulateRound(C, r)
11:   } || {                                      {Coroutine 2: finding a decision}
12:   for scan := 1 to r do
13:      if ∃p_j ∈ Π : state_{scan,j} = "failed" then
14:         Correct := Correct − { p_j }
15:      if ∃p_j ∈ Π : state_{scan,j} = "decided v" then
16:         decide v
17:      else if ∃p_l ∈ Π : state_{scan,j} = "killed" then
18:         add or subtract messages to p_l from faulty processors to have exactly f failures
19:         re-simulate C(k, f) with the new messages to p_l; p_l decides on v
20:         decide v
21:      else if |Correct| ≤ N − f then
22:         select the faulty processor p_l from which all correct
23:                  processors receive a message in round scan
24:         add or subtract messages to p_l from faulty processors to have exactly f failures
25:         re-simulate C(k, f) with the new messages to p_l; p_l decides on v
26:         decide v
27:   }
```

Figure 3: Simulation of algorithm C (code for simulator sim_i)

algorithm $C(k, f)$, an execution of an algorithm in asynchronous shared memory that wait-free solves k-set agreement among $k + 1$ processors, called simulators.

Notice that, in the BG-agreement, a simulator, after taking a snapshot, has a set of candidate winners—the ones that appear in its snapshot. No simulator registering later may win the agreement, and, in general, no simulator registering after any other processor arrived at the wait statement, may win the agreement. Thus, if a simulator, after arriving to the wait statement, observes that all current proposals are the same, this simulator may determine the resolution of the agreement. This means also that a BG-agreement instance is not a black box, but rather an "open" box. Any simulator may access the shared memory used in a particular BG-agreement instance, without invoking BGpropose, e.g., to read all the proposals to this instance and determine the resolution of this instance.

3.1 Overview

We first give here an intuitive idea of the lower bound proof. In the simulation of each synchronous round of the algorithm $C(k, f)$, the $k + 1$ simulators use a series of BG-agreements to decide which messages any processor p_j received and which messages p_j did not receive (this determine the new state of p_j). When a simulator sim_i decides, in any of the BG-agreements, to fail a processor p_j (we also say that p_j

```
1: procedure ResolveInputs()
2:   for each p_j ∈ Π do
3:     BGpropose_{j,0}(i, state_{1,j})
```

Figure 4: Resolving inputs of algorithm C (code for simulator sim_i)

```
1: procedure Execute R_{r,1}
2:   snapshot(state_{r,0}, ..., state_{r,n})
3:   F_{r,i} := { p_j | state_{r,j} ∈ {⊥, "failed", "killed"}} ∪ Suspected_{r,i}
4:   for each p_j ∈ F_{r,i} do
5:     BGpropose_{j,r,1}("failed", state_{r+1,j})
6:   snapshot(state_{r+1,0}, ..., state_{r+1,n})
7:   FinalFaulty_{r+1,i} := { p_j | state_{r+1,j} = "failed" or BGpropose_{j,r,1} has only "failed" proposals}
8:   for each p_j ∈ Correct_{r+1,i} := Π\FinalFaulty_{r+1,i} do
9:     BGpropose_{j,r,2}("p_j receives messages from all processors in Correct_{r+1,i}", state_{r+1,j})
```

Figure 5: First asynchronous phase $R_{r,1}$ (code for simulator sim_i)

was chosen to be failed), this means that sim_i is simulating a run of $C(k, f)$ where processor p_j crashes. The exact simulation performed by simulator sim_i depends on the particular BG-agreement, according to Figure 2. Any simulator sim_i that blocks a BG-agreement does not let the other simulators, involved in the same BG-agreement, decide upon the state of processor p_j; as a simulator may block at most one BG-agreement, then in each round at most k BG-agreements may be unresolved. In the simulation, this is translated into at most k new failures per round of the synchronous run. If a BG-agreement in the "far past" is not resolved, then a simulator is blocked in this BG-agreement, which means that the simulation proceeds from some round on with less than $k + 1$ simulators, and therefore generates less than k failures per round. This, according to condition 1, will force the processors to decide, and allows the simulators to read any processor decision, and then decide on the same value. On the other hand, if no simulator is blocked in any past BG-agreement, then the simulators will identify a processor that is correct and which decides according to condition 2. The simulators may read the decision of this processor, and decide on the same value. The processor that decides will not interfere with the simulation after it decides, because the simulation ensures that this processor fails immediately after deciding. Accomplishing that feat of guaranteeing an a-priori unknown processor to decide according to condition 2 and failing it immediately after deciding, is the crux of the proof.

3.2 Proof

The proof is divided into three parts. The first part inductively explains how a synchronous round r of the algorithm $C(k, f)$, designed for the synchronous message-passing model, may be simulated in the asynchronous shared-memory model. The second part exploits the two conditions of Theorem 1, so that each simulator can reach a decision with the simulation of $C(k, f)$ presented in the first part. The third part shows how to initiate the simulation by instantiating the first part with $r = 1$.

Proof: Assume by contradiction that such an algorithm $C(k, f)$ exists. We will exhibit how $k + 1$ *simulators* $sim_0, \ldots, sim_k$, solve k-set agreement asynchronously in a wait-free manner (i.e., while tolerating k simulator crashes) in shared-memory, using C. This has been proved impossible [3, 14, 17].

Part I: the simulation. The simulators execute 2 asynchronous phases $R_{r,1}$ and $R_{r,2}$ for every synchronous round r of algorithm $C(k, f)$. In the first asynchronous phase $R_{r,1}$ simulating round r of

```
1: procedure Execute R_{r,2}
2:    snapshot(state_{r+1,0}, ..., state_{r+1,n})
3:    snapshot(FinalFaulty_{r+1,0}, ..., FinalFaulty_{r+1,n})
4:    if (i) p_l = ⊥ and
          (ii) ∃sim_q : |FinalFaulty_{r+1,q}| ≥ f and
          (iii) ∄(p_j ∈ Π, r' ≥ 1) : state_{r',j} = "killed" and
          (iv) ∄(p_j ∈ Π, r' ≥ 1) : BGpropose_{r'} has only "kill p_j" proposals then
5:       processorToKill := min_j{ p_j | state_{r+1,j} ∉ {⊥, "failed", "killed"}}
6:       BGpropose_r("kill processorToKill", p_l)
7:    else
8:       BGpropose_r("no processor", p_l)
9:    if p_l ∉ {⊥, "no processor"} then state_{r+1,l} := "killed"
10:   snapshot(state_{r+1,0}, ..., state_{r+1,n})
11:   for all proposed p_j ≠ p_l in BGpropose_r do
12:      Suspected_{r+1,i} := Suspected_{r+1,i} ∪ { p_j }
```

Figure 6: Second asynchronous phase $R_{r,2}$ (code for simulator sim_i)

```
1: procedure SimulateRound(C, r)
2:    execute round r of C using state_{r,0}, ..., state_{r,n}:
3:    • if a processor p_j decides on a value v, then state_{r+1,j} := "decided v"
4:    • otherwise generate the content of the messages to be sent in round r + 1
```

Figure 7: Simulating the code C (code for simulator sim_i)

$C(k, f)$, the simulators will tentatively agree on the state of each processor at the beginning of round $r + 1$ (i.e., on what messages are received by each of the processors in synchronous round r, if any). In the second asynchronous phase $R_{r,2}$ simulating round r of $C(k, f)$, the simulators will tentatively agree on a correct processor, and simulate this processor failure at the beginning of round $r+1$. Asynchronous phases $R_{r,1}$ and $R_{r,2}$ are executed using several BG-agreement instances. The simulation algorithm is shown in Figures 3 to 7, and is detailed hereafter.

In the first asynchronous phase $R_{r,1}$, any simulator sim_i takes a snapshot, and gathers in a set $F_{r,i}$ the processors (a) for which the state at the beginning of round r is not determined, or (b) for which the state at the beginning of round r is determined and indicates that the processor is faulty. Simulator sim_i proposes, in a first series of BG-agreements to determine the state of each of the processors in $F_{r,i}$ at the beginning of round $r + 1$, to fail each processor in $F_{r,i}$. Simulator sim_i, after finishing all these BG-agreements (many of which are possibly unresolved), then takes another snapshot, and gathers in a set $FinalFaulty_{r+1,i} \subseteq F_{r,i}$ the processors in $F_{r,i}$ which are faulty at the beginning of round $r + 1$. (These are the processors decided to fail by resolved BG-agreements, plus the processors in $F_{r,i}$ for which all proposals are to fail them in the corresponding BG-agreement.) For any processor p_j in the complement set $Correct_{r+1,i} = \Pi - FinalFaulty_{r+1,i}$, sim_i obtains the state of p_j at the beginning of round r, either from the first or the second snapshot in $R_{r,1}$. Simulator sim_i writes $FinalFaulty_{r+1,i}$ in shared memory, and then proposes, in a second series of BG-agreements in $R_{r,1}$ to determine the state of each of the processors in $Correct_{r+1,i}$ at the beginning of round $r + 1$, that each processor in $Correct_{r+1,i}$ receives a message from every other processor in $Correct_{r+1,i}$.

Simulator sim_i now moves to the second asynchronous phase $R_{r,2}$; sim_i first takes a snapshot to observe the state of the processors at the beginning of round $r + 1$. In a single BG-agreement to agree

upon a correct processor, sim_i proposes a correct processor, chosen as follows:

1. there is a simulator sim_q in the snapshot taken by sim_i, such that sim_q sees f or more processor failures, and

2. sim_i has not yet chosen a correct processor, nor observed such a processor being chosen, nor guaranteed to be chosen,[3] in a previous asynchronous phase $R_{s,2},\ s < r$.

Otherwise there is no such processor and sim_i proposes a special "no processor" value. The idea is to simulate the failure of the processor agreed upon, at the beginning of round $r + 1$.

Following the BG-agreement of $R_{r,2}$ (not necessarily resolved yet), simulator sim_i takes a snapshot of the proposals to the BG-agreement of $R_{r,2}$, and starts to simulate synchronous round $r + 1$. For each correct processor that appears in the last snapshot taken, that is, a correct processor that may be chosen as the result of the BG-agreement in $R_{r,2}$, its state at the beginning of synchronous round $r + 1$ is not determined until the BG-agreement of $R_{r,2}$ is resolved (we say that the processor is "suspected"). Consequently, in $R_{r+1,1}$, all the simulators propose to fail these processors at the beginning of synchronous round $r + 2$, that is, in the first series of BG-agreements in $R_{r+1,1}$.

Part II: finding a decision. Throughout the simulation, simulator sim_i continuously reads the shared memory in order of increasing rounds starting at round 1, to determine the first processor p_l that has been agreed upon as the result of $R_{r,2}$, for some round r. Because all simulators have the same rule to determine this processor, they will all agree on the same p_l (if one exists). There are two cases however, in which there may never be such a processor:

(i) the simulation goes almost lockstep and less than f processors fail in the simulation, or

(ii) the simulators cannot determine p_l because a past BG-agreement is not yet resolved.

In any of these cases, there will eventually be less than k faulty processors per round. Therefore, the synchronous simulated processors eventually have to decide, according to the algorithm $C(k, f)$.

Now, suppose that none of these cases happen, i.e., every BG-agreement is eventually resolved, but there are forever synchronous rounds with k failures in each round (i.e. the opposite of eventually strictly less than k failures per round). Thus, the number of faulty processors grow without bound as the simulation proceeds far enough. In this case, when reading the shared memory, the simulators will all determine a round m such that m is the first round in which f or more processors are faulty at the beginning of round m. Since for each simulator sim_i each processor in the set $FinalFaulty_{m-1,i}$ is faulty at the beginning of round m, it follows that in round $m - 2$ or less, no correct processor was chosen to fail in $R_{m-2,2}$.[4]

Since at the beginning of round m, there are more than f failures, and at the beginning of round $m - 1$, there are at most f failures, there must be, at the beginning of round m, a processor p_j that fails, and all correct processors at the beginning of round m receive a message from p_j in round $m - 1$. There are two cases:

(i) A correct processor p_l is chosen by the BG-agreement in $R_{m-1,2}$.

(ii) No correct processor is chosen by the BG-agreement in $R_{m-1,2}$.

In the latter case, there necessarily exists at least one simulator sim_i which proposes that nobody be chosen in that BG-agreement, i.e. simulator sim_i observes no other simulator sim_q with $FinalFaulty_{m-1,q} \geq f$. Since sim_i finished all the BG-agreements in $R_{m-1,1}$, no simulator sim_q with $FinalFaulty_{m-1,q} \geq f$

[3] For instance, because a BG-agreement, though not yet resolved, may guarantee that a processor will be chosen, if all propositions are to fail the same processor, and no proposition is to fail no processor.

[4] To see why, suppose by contradiction that a correct processor was chosen to fail at round $m - 2$. Then at least one simulator sim_q has $FinalFaulty_{m-2,q} \geq f$. Since these processors will be faulty at the beginning of $m - 1$, and additionally one correct processor was chosen to fail, there are more than f failures at the beginning of round $m - 1$ contradicting the assumption that m is the first such round.

imposed its proposal in any BG-agreement of $R_{m-1,1}$. Consequently, all processors do not receive messages from at most a set $FinalFaulty_{m-1,j} < f$, for some simulator sim_j. Since there are now more than f faulty processors, the set of faulty processors at the beginning of round m must contain a processor from which all messages are received by correct processors. This processor is chosen to be p_l by all the simulators (ties broken by the lowest processor identifier in case of two such processors.)

In both cases, p_l is a correct processor, and we may add or withdraw enough messages to p_l from other faulty processors in the simulation, to get exactly f failures. (Every simulator can do that in the same deterministic way.) This is possible, since at the beginning of round $m-1$, there are at most f failures.

As round m is the first round in which we choose a correct processor to fail in the second asynchronous phase $R_{m-1,2}$, there are at most k failures per round until round m, as the result of asynchronous simulators being late in a phase. Processor p_l has to decide at the beginning of m, exactly when we fail p_l. Its decision may now be read by all the simulators, which can decide on the same value. This concludes the simulation.

Notice that proposing and chosing a correct processor in one of the second asynchronous phases $R_{r,2}$, in order to simulate its failure, is a transient phenonemon, as a result of the second condition in the choice of p_l. Eventually no processor will be proposed to be faulty after round s for s large enough (in fact in case the number of failures is greater than f then $s = m+1$). Thus, if a simulator is forever late, then eventually the number of failures in each round is less than k since failures occur only because of asynchrony of simulators, and less than $k+1$ simulators proceed thereafter in the simulation.

Part III: starting the simulation. To start the simulation, each simulator proposes in a series of BG-agreements, one for each processor, its simulator identifier as the value proposed by this processor in code $C(k, f)$. Following these BG-agreements, a simulator starts $R_{1,1}$. The initial state of a processor is determined when the corresponding BG-agreement is resolved. □

4 An Early-Deciding Algorithm

Figure 8 presents an early deciding k-set agreement algorithm. For $t < N - k$ (or equivalently, $t \leq n - k$), this algorithm achieves the following bounds: (1) for $0 \leq \lfloor f/k \rfloor \leq \lfloor t/k \rfloor - 2$, every processor that decides, decides by round $\lfloor f/k \rfloor + 2$, and (2) for $\lfloor f/k \rfloor \geq \lfloor t/k \rfloor - 1$, every processor that decides, decides by round $\lfloor f/k \rfloor + 1$. Note that this is a strict generalization of the upper bound on consensus [5, 15],[5] and of (non early deciding) set agreement [7]. For space limitation, we postpone the proof of the algorithm to the optional Appendix.

The algorithm works as follows. Each processor p_i keeps an estimate value est_i, initialized with its proposal value. Processor p_i sends its estimate in every round. At the end of every round, p_i updates est_i with the minimum estimate received from any other processor. Processor p_i also records in $halt_i$ the processors from which it does not receive any message. At the end of any round r, if $|halt_i| < rk$, then the estimate of p_i is a possible decision value. In the next round, p_i sends this estimate with a special DEC decision tag, and decides on its estimate at the end of the round. Any processor p_j that receives a DEC message, adopts the decision value as its new estimate, sends a DEC message in the next round with the decision, and decides on the estimate at the end of that round.

The intuition behind how the algorithm achieves set agreement is as follows. In round r, if p_i observes that $|halt| < rk$, this means that there exists one round $r' \leq r$ where p_i sees at most $k-1$ processor crashes.[6] Hence processor p_i "knows" all but at most $k-1$ values among the smallest values

[5] For uniform consensus, which we consider by default in this paper, the tight lower bound is $f+2$, for $0 \leq f \leq t-2$, and $f+1$, for $f \geq t-1$ [5].

[6] An alternative way to detect the same situation is when p_i sees $k-1$ or less *new* crashes in round r.

```
At processor p_i:
 1: halt := ∅ ; decided := deciding := false
 2: S^r := ∅, 1 ≤ r ≤ ⌊t/k⌋ + 1

 3: procedure propose(v_i)
 4:    est_i := v_i
 5:    for r from 1 to ⌊t/k⌋ + 1 do
 6:       if decided or deciding then send (r, DEC, est_i) to all
 7:       else send (r, EST, est_i) to all
 8:       if deciding then
 9:          decide(est_i) ; return
10:       else if decided then
11:          return
12:       else if received any (r, DEC, est_j) then
13:          est_i := est_j ; deciding := true
14:       else
15:          S^r := {(est_j, j) | (r, EST, est_j) is received in round r from p_j}
16:          halt := Π \ ∪_{(est_j,j)∈S^r} {j}
17:          est_i := min{est_j | (est_j, j) ∈ S^r}
18:          if r = ⌊t/k⌋ and |S^r| ≥ N − k⌊t/k⌋ + 1 then
19:             decided := true ; decide(est_i)
20:          else if |halt| < rk then
21:             deciding := true
22:    decide(est_i)
23:    return
```

Figure 8: An early deciding k-set agreement algorithm (code for processor p_i)

remaining in the system. Processor p_i can thus safely decide on est_i if p_i reaches the end of the next round. (As p_i sends its decision in the next round, we know that every processor that reaches the end of the next round receives p_i's decision if p_i is able to decide.)

We give an intuition of why the algorithm is faster when $\lfloor f/k \rfloor = \lfloor t/k \rfloor - 1$. Note that in this case, every processor that decides, decides by round $\lfloor f/k \rfloor + 1$. At the end of round $\lfloor t/k \rfloor - 1$, the processors have more than k distinct estimate values only if there remain $2k - 1$ processors or less that are still allowed to crash. In round $\lfloor t/k \rfloor - 1$, every processor that detects $k - 1$ or less new crashes may safely decide at the end of round $\lfloor t/k \rfloor$. The reason is the following. First, if $k - 1$ or less processors crash in round $\lfloor t/k \rfloor$, then at most $k - 1$ distinct estimate values remain in the system, and it is safe to decide for any processor. In contrast, if more than $k - 1$ processors crash in round $\lfloor t/k \rfloor$, then $k - 1$ or less processors may still crash. Denote by x the number of processors that detect less than $k - 1$ processor crashes in round $\lfloor t/k \rfloor$. These x processors decide at the end of round $\lfloor t/k \rfloor$. Assume that they immediately crash after deciding. Thus there are at most $k - 1 - x$ processors that may still crash in the last round $\lfloor t/k \rfloor + 1$. At the end of round $\lfloor t/k \rfloor + 1$, at most $k - x$ values may be decided (if $k - 1 - x$ processors crash). In total, processors decide at most on $x + (k - x)$ distinct values.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. In *Proceedings of the* 9^{th} *ACM Symposium on Principles of Distributed Computing*, pages

1–14, 1990.

[2] Y. Afek, G. Stupp, and D. Touitou. Long-lived and adaptive atomic snapshot and immediate snapshot. In *Proceedings of the* 19^{th} *ACM Symposium on Principles of Distributed Computing*, pages 71–80, 2000.

[3] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computation. In *Proceedings of* 25^{th} *ACM Symposium on the Theory of Computing*, pages 91–100. ACM Press, 1993.

[4] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum. The BG distributed simulation algorithm. *Journal of Distributed Computing*, 14:127–146, 2001.

[5] B. Charron-Bost and A. Schiper. Uniform consensus harder than consensus. Technical Report DSC/2000/028, École Polytechnique Fédérale de Lausanne, Switzerland, May 2000.

[6] S. Chaudhuri. More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.

[7] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle. Tight bounds for k-set agreement. *Journal of the ACM*, 47(5):912–943, 2000.

[8] S. Cook. The complexity of theorem proving procedures. In *Proceedings of the* 3^{rd} *ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[9] D. Dolev, R. Reischuk, and H.R. Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, 1990.

[10] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[12] E. Gafni. Round-by-round fault detector–unifying synchrony and asynchrony. In *Proceedings of the* 17^{th} *ACM Symposium on Principles of Distributed Computing*, 1998.

[13] M. Herlihy, S. Rajsbaum, and M. Tuttle. Unifying synchronous and asynchronous message-passing models. In *Proceedings of the* 17^{th} *ACM Symposium on Principles of Distributed Computing*, pages 133–142, 1998.

[14] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[15] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults – a tutorial. Technical report, MIT Technical Report MIT-LCS-TR-821, 2001. (Preliminary version in SIGACT News, Distributed Computing Column, 32(2):45–63, 2001.

[16] N. A. Lynch. *Distributed Algorithms*. Morgan-Kaufmann, 1996.

[17] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: the topology of public knowledge. In *Proceedings of* 25^{th} *ACM Symposium on the Theory of Computing*, pages 101–110, 1993.

Proof of the Algorithm

In the following, we denote the local copy of a variable *var* at processor p_i by var_i, and the value of var_i at the end of round r by var_i^r. $crashed^r$ denotes the set of processors that crash *before* completing round r, $ests^r$ denotes the set of estimate values of every processor at the end of round r. By definition, round 0 ends when the algorithm starts. No processor decides by round 0. We first prove three general claims about the algorithm of Figure 8.

Claim 2 *$ests^r \subseteq ests^{r-1}$.*

Proof: The proof of the claim is straightforward: for any processor p_i, $est_i^r \in ests^{r-1}$. □

Claim 3 *If at the end of round $0 \leq r \leq \lfloor t/k \rfloor$ no processor has decided, and at most l processors crash in round $r+1$, then $|ests^{r+1}| \leq l+1$.*

Proof: Consider that the conditions of the claim hold and assume by contradiction that $|ests^{r+1}| \geq l+2$. By assumption, there are $l+2$ processors with distinct estimate values at the end of round $r+1$. Denote by $q_0, \ldots, q_{l+1}$ these processors, such that $est_{q_i}^{r+1} \leq est_{q_{i+1}}^{r+1}$, for $0 \leq i \leq l+1$. Processors $q_0, \ldots, q_l$ do not send $est_{q_0}^{r+1}, \ldots, est_{q_l}^{r+1}$ in round $r+1$; otherwise, q_{l+1} receives one of the smallest $l+1$ estimate values in round $r+1$. Thus there are $l+1$ processors which send values corresponding to $est_{q_0}^{r+1}, \ldots, est_{q_l}^{r+1}$ in round $r+1$ and which crash in round $r+1$; otherwise, q_{l+1} receives one of the smallest $l+1$ estimate value in round $r+1$. This contradicts our assumption that at most l processors crash in round $r+1$. □

Claim 4 *If, at the end of round $1 \leq r \leq \lfloor t/k \rfloor$, no processor has decided, and $|ests^r| \geq k+1$, then $|crashed^r| \geq rk$.*

Proof: We prove the claim by induction. For the base case $r = 1$, assume that the conditions of the claim hold. That is, at the end of round 1, there exist $k+1$ distinct processors $q_0, \ldots, q_k$ with distinct estimate values. By Claim 3, $|crashed^1| \geq k$. Assume the claim for round $r-1$, and assume the conditions of the claim hold at round r. We prove the claim for round r. By assumption, there are $k+1$ processors $q_0, \ldots, q_k$ at the end of round r with $k+1$ distinct estimates. By Claim 2, $k+1$ processors necessarily reach the end of round $r-1$ with $k+1$ distinct estimates. Thus Claim 4 holds at round $r-1$ (induction hypothesis), and thus, $|crashed^{r-1}| \geq (r-1)k$. By Claim 3, at least k processors crash in round r. Thus $|crashed^r| \geq k + |crashed^{r-1}| \geq rk$. □

The next proposition asserts the correctness of the algorithm.

Proposition 5 *The algorithm in Fig. 8 solves k-set agreement.*

Proof: Validity and Termination are obvious. To prove k-ket agreement, we consider the lowest round r in which some processor decides. Let p_i be one of the processors that decides in round r. We consider three mutually exclusive cases: (1) p_i decides in round $2 \leq r \leq \lfloor t/k \rfloor - 1$, (2) p_i decides in round $r = \lfloor t/k \rfloor$, and (3) p_i decides in round $r = \lfloor t/k \rfloor + 1$. (In the algorithm, no processor decides before round 2.)

Case 1. p_i necessarily decides at line 9, and thus executes line 21 in round $r-1$, where *deciding* is set to *true*. (Because no processor decides before p_i, p_i may not receive any DEC message before deciding; and because $r \leq \lfloor t/k \rfloor - 1$, p_i may not decide at line 19.) In round $r-1$, p_i executes line 21 only if p_i

evaluates $|crashed^{r-1}| < rk$ at line 20. Thus, from Claim 4, there are at most k distinct estimates at the end of round $r-1$, which ensures agreement.

Case 2. There are two cases to consider: (1) p_i decides at line 9, after executing line 21 at the end of round $r-1$, or (2) p_i decides at line 19. (Because no processor decides before p_i, p_i may not receive any DEC message before deciding.) In case (1), p_i executes line 21 in round $r-1$ only if p_i evaluates $|crashed^{r-1}| < rk$ at line 20. Thus, from Claim 4, there are at most k distinct estimates at the end of round $r-1$, which ensures agreement. In case (2), we consider $ests^{r-1}$. If $|ests^{r-1}| \leq k$, agreement is ensured thereafter. Thus consider that $|ests^{r-1}| \geq k+1$. By Claim 4, there exist $k+1$ distinct processors with different estimates at the end of round $r-1$ only if $|crashed^{r-1}| \geq k(r-1) = k(\lfloor t/k \rfloor - 1) \geq t-2k+1$, or, equivalently, only if at most $2k-1$ processors may crash in the two subsequent rounds (rounds $\lfloor t/k \rfloor$ and $\lfloor t/k \rfloor + 1$). In round $\lfloor t/k \rfloor$, p_i decides at line 19 only if p_i receives at least $n - k\lfloor t/k \rfloor + 1$ messages. Thus, by Claim 3, the processors that decide at the end of round $\lfloor t/k \rfloor$, including p_i, decide on at most k distinct values. Denote by x the number of processors that effectively crash in round $\lfloor t/k \rfloor$, and by y the number of processors that decide at the end of round $\lfloor t/k \rfloor$. We distinguish two cases: (a) $x \leq k-1$, and (b) $x \geq k$. In case (a), by Claim 3, $k-1$ values or less remain in the system at the end of round $\lfloor t/k \rfloor$; agreement is then ensured. In case (b), at most $2k-1-x \leq k-1$ processors may crash among the processors that decide at the end of round $\lfloor t/k \rfloor$ and the processors that take part to round $\lfloor t/k \rfloor + 1$. We claim that the total number of distinct decision values is at most k. Indeed, denote by y_{crash} the number of processors that decide at the end of round $\lfloor t/k \rfloor$ and then immediately crash. In round $\lfloor t/k \rfloor + 1$, at most $k - 1 - y_{crash}$ may crash. By Claim 3 processors that decide at the end of round $\lfloor t/k \rfloor + 1$ may decide on at most $k - y_{crash}$ distinct estimate values. Hence the maximum number of decided values is $(k - y_{crash}) + y_{crash} = k$.

Case 3. By contradiction, consider that, at the end of round $\lfloor t/k \rfloor + 1$, there exist $k+1$ distinct processors $q_0, \ldots, q_k$ with different estimates, and which decide on their estimates. By Claim 2, there exist $k+1$ processors with distinct estimates at the end of round $r-1$. By Claim 4 and because $r = \lfloor t/k \rfloor + 1$, $|crashed^{r-1}| > k(r-1) = k\lfloor t/k \rfloor > t-k$. By Claim 3, there exist k processors that crash in round $\lfloor t/k \rfloor + 1$. Thus $|crashed^{r}| \geq k + |crashed^{r-1}| = k + k\lfloor t/k \rfloor > t$. A contradiction. □

The next proposition asserts the efficiency of the algorithm.

Proposition 6 *In any run with $0 \leq f \leq t$ failures, any processor that decides, decides*

1. *by round $\lfloor f/k \rfloor + 2$, if $0 \leq \lfloor f/k \rfloor \leq \lfloor t/k \rfloor - 2$, and*
2. *by round $\lfloor f/k \rfloor + 1$, if $\lfloor f/k \rfloor \geq \lfloor t/k \rfloor - 1$.*

Proof: We proceed by separating both cases.

Case 1. Assume a run with f failures, such that $\lfloor f/k \rfloor \leq \lfloor t/k \rfloor - 2$. By contradiction, assume that there exists a processor p_i for which $|halt_i^r| \geq rk$, for $r = \lfloor f/k \rfloor + 1$. (If $|halt_i^r| < rk$, then p_i decides at line 9 in the next round.) Processor p_i does not decide in round r; in particular, p_i does not receive any DEC message in round r. We have $|halt_i^r| \geq rk = (\lfloor f/k \rfloor + 1)k = \lfloor f/k \rfloor k + k > f$. A contradiction.

Case 2. Assume a run with f failures, such that $\lfloor f/k \rfloor \geq \lfloor t/k \rfloor - 1$. First assume that $\lfloor f/k \rfloor = \lfloor t/k \rfloor - 1$, and assume by contradiction that there exists a processor p_i that does not decide by round $r = \lfloor f/k \rfloor + 1$. Thus p_i does not receive any DEC message in round r. Assume by contradiction that p_i does not decide at line 19. Thus $|S^r| < N - k\lfloor t/k \rfloor + 1$, and $f > k\lfloor t/k \rfloor - 1$. This implies in turn that $\lfloor f/k \rfloor > \lfloor t/k \rfloor - 1$. A contradiction. When $\lfloor f/k \rfloor = \lfloor t/k \rfloor$, then any processor that decides, decides by round $\lfloor f/k \rfloor + 1 = \lfloor t/k \rfloor + 1$. □