Of Choices, Failures and Asynchrony: The Many Faces of Set Agreement

Dan Alistarh[1], Seth Gilbert[1], Rachid Guerraoui[1], and Corentin Travers[2]*

[1] Swiss Federal Institute of Technology, Lausanne, Switzerland
[2] Technion, Haifa, Israel

Abstract. Set agreement is a fundamental problem in distributed computing in which processes collectively choose a small subset of values from a larger set of proposals. The impossibility of fault-tolerant set agreement in asynchronous networks is one of the seminal results in distributed computing. The complexity of set agreement in synchronous networks has also been a significant research challenge. Real systems, however, are neither purely synchronous nor purely asynchronous. Rather, they tend to alternate between periods of synchrony and periods of asynchrony.

In this paper, we analyze the complexity of set agreement in a such a "partially synchronous" setting, presenting the first (asymptotically) tight bound on the complexity of set agreement in such systems. We introduce a novel technique for simulating, in fault-prone asynchronous shared memory, executions of an asynchronous and failure-prone message-passing system in which some fragments appear synchronous to some processes. We use this technique to derive a lower bound on the round complexity of set agreement in a partially synchronous system by a reduction from asynchronous wait-free set agreement. We also present an asymptotically matching algorithm that relies on a distributed asynchrony detection mechanism to decide as soon as possible during periods of synchrony.

By relating environments with differing degrees of synchrony, our simulation technique is of independent interest. In particular, it allows us to obtain a new lower bound on the complexity of early deciding k-set agreement complementary to that of [12], and to re-derive the combinatorial topology lower bound of [13] in an algorithmic way.

1 Introduction

Set agreement was first introduced by Chaudhuri [6] to capture the power of allowing more choices than *consensus* [16], where only a single decision value is permitted. Each process p_i begins with an initial value v_i; eventually, every process outputs one of the initial values as a decision. In k-set agreement, the set of all values output can be of size at most k. When $k = 1$, set agreement reduces to *consensus*. When $k = n$, the problem is trivial, i.e., processes can act entirely independently.

* Supported in part by a Sam & Cecilia Neaman Fellowship.

In a collection of seminal papers, Borowsky, Gafni, Herlihy, Saks, Shavit, and Zaharoglou [5, 14, 17] showed that fault-tolerant asynchronous set agreement is impossible (while at the same time revealing a deep connection between distributed computing and algebraic topology). Chaudhuri et al. [7] further developed these techniques, establishing a tight lower bound on the round complexity of *synchronous* set agreement: in a system with t failures, at least $\lfloor t/k \rfloor + 1$ rounds are necessary. More recently, Gafni et al. [12] and Guerraoui et al. [13] considered the feasibility of reaching an *early decision*: how fast can an algorithm tolerating up to t failures decide in an execution with at most $f < t$ failures? They both show, in different ways, that at least $\lfloor f/k \rfloor + 2$ rounds are needed.

Set agreement has been extensively studied in both synchronous and asynchronous systems. Real world distributed systems, however, are neither purely synchronous nor purely asynchronous. Instead, they tend to exhibit periods of synchrony when the network is well behaved, and periods of asynchrony when the network is poorly behaved. To describe such a system, Dwork et al. [9] introduced the idea of *partial synchrony.* They assume for every execution some (unknown) time GST (*global stabilization time*), after which the system is synchronous. In this paper, we study the *feasibility* and *complexity* of set agreement in the context of partially synchronous systems, determining the minimum-sized window of synchrony in which k-set agreement can be solved. Of course, the lower bounds for synchronous systems [7, 10] imply an immediate lower bound here of $\lfloor \frac{t}{k} \rfloor + 1$ rounds. The question, then, is whether there exists any matching algorithm that terminates in a synchronous window of size $\lfloor \frac{t}{k} \rfloor + 1$, or is there some inherent cost to tolerating asynchrony? Moreover, how does this cost depend on t and k?

We answer these questions by showing that at least $\lfloor \frac{t}{k} \rfloor + 2$ synchronous rounds are required for k-set agreement, and demonstrating an algorithm that terminates in any window of synchrony of size at least $\lfloor \frac{t}{k} \rfloor + 4$ rounds. Together, these results tightly bound the inherent price of tolerating some asynchrony.

Lower Bound By Reduction. The technique for deriving the lower bound is an important contribution in end of itself, as it provides new insights into the complexity of set agreement. Instead of relying on topology, as is typically required for set agreement lower bounds, we derive our result by reducing the feasibility of asynchronous set agreement to the problem of solving set agreement in a window of size $\lfloor \frac{t}{k} \rfloor + 1$. Since asynchronous set agreement is known to be impossible, this reduction immediately implies that at least $\lfloor \frac{t}{k} \rfloor + 2$ synchronous rounds are required for k-set agreement.

Early Deciding Synchronous Set Agreement. Our technique turns out to be of more general interest, as we can re-derive and extend existing lower bounds for synchronous *early deciding* set agreement. It has been previously shown [12, 13] that even in an execution with $f < t$ failures, some process cannot decide prior to round $\lfloor f/k \rfloor + 2$.

Using our simulation technique, we re-derive both previous lower bounds in a simpler and more general manner, in the standard model where t and n are bounded and known. Of note, both lower bounds are corollaries of a single theorem that relates the number of processes which decide early with the worst-case round complexity of an algorithm. Basically, we show that if d processes decide by round $\lfloor f/k \rfloor + 1$ in executions with at most f failures, then in the worst-case, some process takes at least time $\lfloor t/k \rfloor + E(\cdot) + 1$ to decide (where E is a function of t, k and d). Due to space limitations, we leave the presentation of these results to the full version of the paper [2].

Upper Bound for Eventually Synchronous Agreement. We then present the first known algorithm for set agreement that tolerates periods of asynchrony. Our algorithm guarantees correctness, regardless of asynchrony, and terminates as soon as there is a window of synchrony of size $\lfloor t/k \rfloor + O(1)$. For simplicity, we show synchronous round complexity of $\lfloor t/k \rfloor + 4$; a tighter analysis in the full version yields $\lfloor t/k \rfloor + 3$ for $t \geq k^2$. Note that a previous paper [3] presents an algorithm that exactly matches the lower bound, for the special case of *consensus* ($k = 1$). Thus, closing the one round gap between the two bounds remains an intriguing challenge.

Two basic ideas underlie our algorithm. First, processes collectively execute an *asynchrony detection* sub-protocol that determines whether a round appears synchronous or asynchronous. A process can decide when it sees $\lfloor t/k \rfloor + O(1)$ synchronous rounds. Even so, different sets of processes may have different views of the actual execution when the decision occurs, since there are only $\lfloor t/k \rfloor + O(1)$ rounds to exchange information. Second, each process maintains an *estimate*, i.e., a value that it is leaning toward choosing. In each round, each process adopts the minimum estimate that it receives. If a process is about to decide, however, it can *elevate the priority* of its estimate, causing other processes to adopt its value instead.

Implications. Several implications arise from our simulation technique and its usage. First, it provides additional evidence that the impossibility of fault-tolerant asynchronous k-set agreement is a central result in distributed computing, as it implies non-trivial results in both partially synchronous and synchronous models. Second, it highlights close connections between models that have differing levels of synchrony. In particular, our simulation technique takes advantage of structural similarities between *eventually synchronous* set agreement and *early deciding* set agreement to establish lower bounds in two different models of synchrony. The uncertainty regarding asynchrony (found in a partially synchronous execution) turns out to be fundamentally similar to the uncertainty regarding failures (found in an early deciding execution).

2 Model

In this section, we define three basic models of computation. The *partially synchronous model* $ES_{n,t}$ consists of n deterministic processes $\Pi = \{p_1, \ldots, p_n\}$, of which up to $t < n$ may fail by crashing. (Note that the algorithm in Section 4 uses $t < n/2$.) The processes communicate via a message-passing network, modeled much as in [8,9,15]: time is divided into *rounds*; however, there is no assumption that every message broadcast in a round is also delivered in that round. Instead, we assume only that if all non-failed processes broadcast a message in round r, then each process receives at least $n - t$ messages in that round[3]. We assume that the network is *partially synchronous*: there is some round GST after which every message sent by a non-failed process is delivered in the round in which it is sent. The *synchronous model* $S_{n,t}$ is identical to $ES_{n,t}$, except that we assume every process knows, *a priori*, that $GST = 0$, i.e., that every message is delivered in the round that it is sent.

The *asynchronous model* $AS_{n,k}$ consists of n processes Π, up to k of which may crash. The processes communicate via single-writer, multi-readers (SWMR) registers. The memory is organized in arrays $X[1..n]$ of n registers; entry $X[i]$ of an array can be written only by p_i. In addition to $\mathsf{read}()$ and $\mathsf{write}()$ operations, a process can also invoke $X.\mathsf{snapshot}()$ to read all the contents of X in a logically instantaneous single operation. Let x and x' be the result of any two $\mathsf{snapshot}$ operations on X. We assume that the following hold: *Containment:* $x \subseteq x' \vee x' \subseteq x$; *Self inclusion:* Let v be the value written by p_i in $X[i]$ prior to invoking $X.\mathsf{snapshot}()$, with no intervening $x.\mathsf{write}(\cdot)$ operations; let x be the result of the snapshot operation; then $x[i] = v$. Implementation of snapshot on top of SWMR registers can be found in [1, 4], and thus they provide no extra power. k-set agreement is impossible in $AS_{n,k}$ [5, 14, 17].

3 Simulating Synchronous Views: a Lower Bound for *k*-Set Agreement

In this section, we present an algorithm for simulating, in the asynchronous model $AS_{n,k}$, executions in $ES_{n,t}$ of a k-set agreement algorithm $\mathcal{A}$. The simulation pseudocode is presented in Figure 1. Each process in $AS_{n,k}$ simulates one process executing $\mathcal{A}$ in $ES_{n,t}$. (We refer to the processes in $AS_{n,k}$ as *simulators.*) Each execution e simulated by our algorithm is a valid execution of $\mathcal{A}$ in model $ES_{n,t}$, and satisfies the following property: at least one process, whose simulator is correct, observes a window of synchrony of length at least $\lfloor t/k \rfloor + 1$. More precisely, for each simulated execution e, there exists a round R and a process p such that p cannot distinguish, by the end of round $R + \lfloor t/k \rfloor + 1$, execution e from some execution e' in which the global stabilization round GST is equal to R. We say that p has a *synchronous view* of length $\lfloor t/k \rfloor + 1$.

Our simulation relies on the ideas introduced in [10]. The goal in [10] is to simulate, in $AS_{n,k}$, executions of the synchronous model $S_{n,t}$. The simulation ensures that (1) whenever a message is not delivered by some process, the sender is simulated as failed in every following round and, (2) in each round, at most k new failures occur. The first property guarantees that the simulated execution is synchronous, while the second property implies that up to $\lfloor t/k \rfloor$ rounds can be simulated without exceeding the maximum number of failures t allowed by the model $S_{n,t}$.

We observe that in an execution of $\lfloor t/k \rfloor + 1$ rounds simulated with the technique described in [10], although more than t processes might have failed in the simulated execution, at least one process p observes no more than t failures and

[3] This can be implemented by delaying a round $r+1$ message until at least $n-t$ round r messages have been received.

still perceives the execution as synchronous. Thus, if we assume a k-set agreement protocol for model $ES_{n,t}$ where every process decides by the end of round $GST + \lfloor t/k \rfloor + 1$, process p must obtain a decision. If its associated simulator does not fail, the decision can be propagated to every simulator which allows them to decide. In the other case, we repeat the simulation for another $\lfloor t/k \rfloor + 1$ rounds, again resulting in a process either deciding or its associated simulator failing. Eventually, after $k+1$ repetitions (which we refer to as "phases"), we argue that some process decides and does not fail.

3.1 Basic Setup

The simulation depends on three parameters: the algorithm $\mathcal{A}$ being simulated, the number of phases $numP$, and an array $R_1, R_2, \ldots, R_{numP+1}$ where each R_i is the first round in the i^{th} phase.

For process p_i, the algorithm $\mathcal{A}$ is described by a function $\mathsf{compute}(r, rec)$, where r is a round number and rec a set of messages received by p_i in round r. The $\mathsf{compute}$ function returns a pair (d_i, m_i), where m_i is the message to be sent in the next round, and d_i is the decision value or $\perp$. Without loss of generality, we assume that each process sends the same message to all other processes.

3.2 Simulating Synchronous Rounds

Each process in $AS_{n,k}$ simulates one process executing $\mathcal{A}$ in $ES_{n,t}$. The simulation begins with a call to $\mathsf{propose}(v_i)$ (line 5), where v_i is p_i's proposal.

The simulation is divided into *phases* (lines 8–13): in each round of a phase in which no simulators fail, there is at least one process that sees the phase as a *window of synchrony* within a (possibly) asynchronous execution.

Round overview. In order to simulate round r (lines 11–13), process p_i invokes $\mathsf{simulate}(m_i, r)$ (line 12), where m_i is its message for round r, which was computed previously. The $\mathsf{simulate}$ procedure returns pairs $\langle j, m_j \rangle$, where m_j is the round r message "sent" by p_j. The simulator then calls the $\mathsf{compute}$ function (line 13), which returns d_i, a possible decision, and m_i, the next message to send. If $d_i \neq \perp$, simulator p_i writes the decision value d_i in the shared array DEC before deciding that value (line 13). Similarly, if a simulator observes that a value $\neq \perp$ has been written in DEC, it decides that value (lines 14–16).

Simulating a round. The $\mathsf{simulate}$ function (lines 17–30) carries out the send/receive step. For round r, simulator p_i writes the message m_i into the register $VAL[r][i]$ (line 18), and then performs repeated snapshots of $VAL[r]$ (line 19) to discover the messages of other simulators. Since k simulators may fail in $AS_{n,k}$, the simulator cannot wait for messages from all n simulators. As soon as p_i discovers $n-k$ messages in its snapshot of $VAL[r]$, it continues. The variable M_i stores the set of up to k processes from which some message was missed. The simulators then need to agree on which messages to deliver in the simulated round and which messages were missed; if a message is missed from some process p_j, then the simulators collectively "fail" process p_j in all future simulated rounds.

Adopt-commit objects. The simulators use *adopt-commit* objects (introduced in [10, 18]) to coordinate which processes have "failed" in the simulation.

```
Parameters: $\mathcal{A}$, $numP$, $[R_1, \ldots, R_{numP+1}]$
Shared variables:
  $AC[1..R_{numP+1}][1..n]$, array of adopt-commit objects
  $DEC[1..n]$, $VAL[1..R_{numP}+1][1..n]$, array of SWMR registers.
procedure propose($v_i$): start Task T1; start Task T2;
Task T1:
  $(\_, m_i) \leftarrow \mathsf{compute}(0, v_i, \mathsf{true})$ % messages for the first round
  for $\rho = 1$ to $numP$ do
      % Begin a new phase:
      $S_i \leftarrow \emptyset$
      for $r = R_\rho$ to $R_{\rho+1} - 1$ do
          $rec_i \leftarrow \mathsf{simulate}(m_i, r)$ % Simulate send/receive of round $r$.
          $(d_i, m_i) \leftarrow \mathsf{compute}(r, rec_i)$ % Compute message for next round. if
          $d_i \neq \perp$ then $DEC[i].\mathsf{write}(d_i)$; stop T2; return $d_i$
Task T2:
  repeat for $j = 1$ to $n$ do $dec_i[j] \leftarrow DEC[i]$ until $(\exists \ell : dec_i[\ell] \neq \perp)$
  stop T1; return $dec_i[\ell]$
procedure simulate( $m_i$, $r$ ) % Simulate round $r$ where $p_i$ sends message $m_i$.
  $rec_i \leftarrow \emptyset$; $VAL[r][i].\mathsf{write}(m_i)$
  repeat $view_i \leftarrow VAL[r].\mathsf{snapshot}()$ until $|\{j : view_i[j] = \perp\}| \leq k$
  $M_i \leftarrow \{j : view_i[j] = \perp\}$
  for $j = 1$ to $n$ do
      if $j \in S_i \cup M_i$ then $state_i[j] \leftarrow AC[r][j].\mathsf{propose}(\text{suspect})$
      else $state_i[j] \leftarrow AC[r][j].\mathsf{propose}(\text{alive})$
      if $state_i[j] = (commit, suspect\ )$ then $S_i \leftarrow S_i \cup \{j\}$
      else if $state_i[j] = (adopt, suspect\ )$ then $S_i \leftarrow S_i \cup \{j\}$;
      $rec_i \leftarrow rec_i \cup \{\langle j,\ VAL[r][j]\rangle\}$
      else $rec_i \leftarrow rec_i \cup \{\langle j,\ VAL[r][j]\rangle\}$
  % Complete view of round $r$, if necessary:
  if $|rec_i| < n - t$ then $rec_i \leftarrow \{\langle j, view_i[j]\rangle : view_i[j] \neq \perp\}$ ;
  if $\langle i, m_i\rangle \notin rec_i$ then $rec_i \leftarrow rec_i \cup \{\langle i, view_i[i]\rangle\}$ ;
  return $rec_i$
```

Fig. 1. Simulating $\mathcal{A}$ in $AS_{n,k}$, code for simulator p_i.

An adopt-commit object AC is invoked via a $\mathsf{propose}(v)$ operation, and returns a decision (dec, v) where $dec \in \{\mathsf{adopt}, \mathsf{commit}\}$. The object satisfies the following properties: *1. Termination:* Each invocation by a correct process terminates. *2. Validity:* If a process decides (dec, v) then some process invoked $AC.\mathsf{propose}(v)$. *3. Agreement:* If a process decides (commit, v), then every decision is $(\cdot, v)$. *4. Convergence:* If every process proposes the same v, then (commit, v) is the only possible decision. Implementations of adopt-commit objects in $AS_{n,k}$ can be found in [10, 18]. These implementations also satisfy: *5. Commit Validity:* Assume p_j invokes $AC.\mathsf{propose}(v)$; then p_j cannot get back (commit, v') with $v \neq v'$.

Agreeing on failures. After completing the snapshots, the simulators use the adopt-commit objects to agree on which processes have failed. Simulator p_i stores in S_i a set of suspected processes, and it resets S_i to $\emptyset$ at the beginning of each phase. Throughout the phase, processes are added to S_i based on the output of the adopt-commit objects. If a process p_i misses a message from a process p_j in round r (i.e., if $p_j \in M_i$), or if process p_i suspects p_j (i.e., if $p_j \in S_i$), then its simulator proposes suspecting p_j using $AC[r][j]$ (line 22). Otherwise, the simulator proposes that p_j is *alive* (line 23).

There are three possible decisions. First, (commit, *suspect*) (line 24): in this case, the simulation fails process p_j in round r. By *agreement*, we know that every simulator either adopts or commits to suspecting p_j, and so process i adds p_j to S_i. Second, (adopt, *suspect*) (line 25): in this case, we cannot determine whether p_j is failed or not in round r; even so, to be safe, simulator p_i adds p_j to S_i. We know, however, by *validity* that some process proposed p_j as alive, and so we know that $VAL[r][j]$ contains the message from p_j, which we add to the set rec_i of messages to deliver. Finally ($\cdot$, *alive*) (line 26): as in the second case, we add the message from $VAL[r][j]$ to rec_i. Notice that if any simulator commits to failing p_j, then every other simulator will either adopt or commit to failing p_j and add p_j to S_i. By *convergence*, in the following round, every simulator commits to failing p_j. By using the adopt-commit objects in this way, we ensure that simulated views remain synchronous.

The end of the phase. This approach results in simulating up to k new failures in each round. Eventually, the number of simulated failures may surpass t, the bound on failures in $ES_{n,t}$. Consequently, for some processes, the set of messages rec_i may no longer contain an appropriate set of messages to deliver. In that case, simulator p_i augments the set rec_i to ensure that it contains enough messages ($|rec_i| \geq n - t$, line 28) and that it contains the round r message of p_i (line 29).

Therefore, not all processes may maintain a synchronous view. However, we can show that if the length of the phase is at most $\lfloor t/k \rfloor + 1$ rounds, at least one process is able to maintain its synchronous view through the end of a phase.

3.3 Lower Bound on Set Agreement in $ES_{n,t}$

We now show how to use the simulation technique to prove a lower bound on set agreement in $ES_{n,t}$. We begin, for the sake of contradiction, by assuming that algorithm $\mathcal{A}$ solves k-set agreement in $ES_{n,t}$ in any window of synchrony of size $\lfloor t/k \rfloor + 1$. The simulation uses $k+1$ phases, each of length $\lfloor t/k \rfloor + 1$, i.e., $R_\rho = (\rho - 1)(\lfloor t/k \rfloor + 1) + 1$. We show that the resulting simulation of $\mathcal{A}$ solves k-set agreement in $AS_{n,k}$, which is known to be impossible, implying that no such algorithm $\mathcal{A}$ exists. This implies that any k-set agreement protocol requires at least $\lfloor t/k \rfloor + 2$ synchronous rounds to decide.

First, we list some of the properties of the simulation, whose proofs can be found in the full version of the paper [2]: *(P1)* the simulated execution of $\mathcal{A}$ is a valid execution of $\mathcal{A}$ in $ES_{n,t}$; *(P2)* each phase ρ appears synchronous: if a process sees $f \leq t$ failures by the end of round r, then it perceives the first r rounds of phase ρ as synchronous; *(P3)* there are at most t simulated failures at the beginning of the last simulated round in phase ρ; *(P4)* some simulated process sees no new failures in the last round. Since each phase is of length $\lfloor t/k \rfloor + 1$, and since $\mathcal{A}$ guarantees a decision in a window of synchrony of size $\lfloor t/k \rfloor + 1$, it follows from properties *(P1)–(P4)* that by the end of phase ρ, a process either decides, having seen the entire phase as synchronous, or its associated simulator fails.

Lemma 1. *For every phase ρ, if no process decides and writes its decision to DEC prior to the end of phase ρ, then at least one process that begins phase ρ fails before beginning phase $\rho + 1$.*

We conclude that our simulation of algorithm $\mathcal{A}$ solves k-set agreement in $AS_{n,k}$. *Agreement* follows from the fact that our simulation is a valid simulation of $\mathcal{A}$ in $ES_{n,t}$, and *termination* follows from the fact that if there is no decision, then at least one simulator fails in every phase; since there are only k possible failures in $AS_{n,k}$, by the end of phase $k + 1$, some process must decide.

Lemma 2. *The algorithm in Figure 1 simulating $\mathcal{A}$ solves k-set-agreement in $AS_{n,k}$.*

Since k-set agreement is impossible in $AS_{n,k}$, we conclude:

Theorem 1. *There is no algorithm $\mathcal{A}$ for $ES_{n,t}$ that decides by round $GST + \lfloor t/k \rfloor + 1$, i.e., within a window of synchrony of size $\lfloor t/k \rfloor + 1$.*

4 k-Set Agreement Algorithm for $ES_{n,t}$

We present an algorithm named K4 which solves k-set agreement in a window of synchrony of size $\lfloor t/k \rfloor + 4$. The algorithms requires a majority of correct processes, i.e., $t < n/2$. The pseudocode can be found in Figure 2. The proof of correctness for the protocol can be found in the full version of this paper [2].

4.1 Description

K4 is a round-based full-information protocol. Each process maintains a local estimate est_i, representing its preferred decision, and sets $Active_i$ and $Failed_i$, which denote the processes that p_i believes to be alive and failed (respectively). In every round, each process broadcasts its entire state (line 5), and receives all the messages for the current round (line 6), updating its view of which processes have failed and which rounds are synchronous (lines 7–10). A process decides if it receives a message from another process that has already decided (lines 11–12), or if it sees $\lfloor t/k \rfloor + 4$ consecutive synchronous rounds (line 13). If no decision is reached, then the estimate est_i is updated in lines 15–17. There are two key components to K4: accurately determining whether rounds are synchronous (which is critical for ensuring liveness), and updating the estimate (which is critical for ensuring agreement).

Detecting Asynchrony. updateSynchDetector() merges information into the *Active* and *Failed* sets; if a process believes that p_ℓ was active in round r, then p_ℓ is added to $Active[r]$; if it believes that p_ℓ was failed during round r, then p_ℓ is added to $Failed[r]$ (see lines 20–23). It then determines based on $Active[r]$

```
1  procedure propose($v_i$)$_i$
2    $est_i \leftarrow v_i$; $r_i \leftarrow 1$; $msgSet_i \leftarrow \emptyset$; $sFlag_i \leftarrow$ false
3    $Active_i \leftarrow [\,]$; $Failed_i \leftarrow [\,]$; $AsynchRound_i \leftarrow [\ ]$
4    while true do
5      send( $est_i$, $r_i$, $sFlag_i$, $Active_i$, $Failed_i$, $AsynchRound_i$, $decide_i$ ) to all
6      wait until received at least $(n - t)$ messages for round $r_i$
7      $msgSet_i[r_i] \leftarrow$ messages that $p_i$ receives in round $r_i$
8      $Active_i[r_i] \leftarrow$ processes from which $p_i$ gets messages in round $r_i$
9      $Failed_i[r_i] \leftarrow \Pi \setminus Active_i[r_i]$
10     updateSynchDetector() % Update the state of $p_i$.
11     if ( $\exists msg_p \in msgSet_i$ with $msg_p.decided_p =$ true )  then
12       $decide_i \leftarrow$ true; $est_i \leftarrow msg_p.est_p$
13     if $(sCount_i = \lfloor t/k \rfloor + 4)$ then  $decide_i \leftarrow$ true
14     if $(decided_i =$ false$)$  then
15       $flagProcs_i \leftarrow \{\ p \in Active_i[r_i] \mid sFlag_p =$ true $\}$
16       if  $flagProcs_i \neq \emptyset$ then  $est_i \leftarrow \min_{q \in flagProcs_i}(est_q)$
17       else  $est_i \leftarrow \min_{q \in Active_i[r_i]}(est_q)$
18     $r_i \leftarrow r_i + 1$
19 procedure updateSynchDetector()
20   for every $msg_j \in msgSet_i[r_i]$ do
21     for round $r$ from $1$ to $r_i - 1$ do
22       $Active_i[r] \leftarrow msg_j.Active_j[r] \cup Active_i[r]$
23       $Failed_i[r] \leftarrow msg_j.Failed_j[r] \cup Failed_i[r]$
24   for round $r$ from $1$ to $r_i - 1$ do
25     $AsynchRound_i[r] \leftarrow$ false
26     for round $k$ from $r + 1$ to $r_i$ do : if $(Active_i[k] \cap Failed_i[r] \neq \emptyset)$ then
       $AsynchRound_i[r] \leftarrow$ true
27   $sFlag_i \leftarrow$ false
28   $sCount_i \leftarrow \mathbf{max}_\ell(\forall r' \in [r_i - \ell, r_i], AsynchRound_i[r'] =$ false$)$
29   if  $sCount_i = \lfloor t/k \rfloor + 3$ then  $sFlag_i \leftarrow$ true
```

Fig. 2. The K4 algorithm, at process p_i.

and *Failed*$[r]$ sets whether round r seems synchronous (lines 24-26). A round r is deemed asynchronous if some process p_ℓ is believed to have failed in round r (i.e., $p_\ell \in Failed[r]$), and yet is also believed to be alive at some later round $k > r$ (i.e., $p_\ell \in Active[k]$). Finally, process p_i sets a flag *sFlag* to true if it sees the previous $\lfloor t/k \rfloor + 3$ rounds as synchronous (line 29).

Updating the estimate. Each process updates the estimate in every round. Estimates have two levels of priority: if a process has seen $\lfloor t/k \rfloor + 3$ synchronous rounds, i.e., if it is "ready to decide," then its estimate has high priority; other estimates are awarded normal priority. A process adopts the minimum prioritized estimate, if one exists (line 16); otherwise, it adopts the minimum estimate received in the current round (line 17).

5 Conclusion

We have presented a new technique for simulating synchronous and partially synchronous executions in asynchronous shared memory. Our technique allows us to characterize the complexity of set agreement in partially synchronous systems, as well as to refine earlier lower bounds for early-deciding synchronous set agreement. One direction of future work is to extend our lower bound results to other tasks by encapsulating the Extended BG simulation [11]. On the algorithmic side, the solvability of set agreement in partially synchronous systems without a majority of correct processes remains an open question.

References

1. Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, 1993.
2. D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Of choices, failures and asynchrony: The many faces of set agreement. Technical report, http://lpd.epfl.ch/alistarh/kset-ps/kset-TR-full.pdf.
3. D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. How to solve consensus in the smallest window of synchrony. In *DISC*, pages 32–46, 2008.
4. H. Attiya and O. Rachman. Atomic snapshots in $o(n \log n)$ operations. *SIAM J. Comput.*, 27(2):319–340, 1998.
5. E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100, 1993.
6. S. Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Inf. Comput.*, 105(1):132–158, 1993.
7. S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle. A tight lower bound for k-set agreement. In *FOCS*, pages 206–215. IEEE, 1993.
8. P. Dutta and R. Guerraoui. The inherent price of indulgence. *Distributed Computing*, 18(1):85–98, 2005.
9. C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.
10. E. Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract). In *PODC*, pages 143–152, 1998.
11. E. Gafni. The Extended BG simulation and the characterization of t-resiliency. In *STOC*, 2009.
12. E. Gafni, R. Guerraoui, and B. Pochon. From a static impossibility to an adaptive lower bound: the complexity of early deciding set agreement. In *STOC*, pages 714–722, 2005.
13. R. Guerraoui, M. Herlihy, and B. Pochon. A topological treatment of early-deciding set-agreement. In *OPODIS*, pages 20–35, 2006.
14. M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.
15. I. Keidar and A. Shraer. Timeliness, failure-detectors, and consensus performance. In *PODC*, pages 169–178, 2006.
16. L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ToPLaS*, 4(3):382–401, 1982.
17. M. E. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.
18. J. Yang, G. Neiger, and E. Gafni. Structured derivations of consensus algorithms for failure detectors. In *PODC*, pages 297–308, 1998.