Algorithmica (2012) 62:595–629
DOI 10.1007/s00453-011-9581-7

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

Dan Alistarh · Seth Gilbert · Rachid Guerraoui · Corentin Travers

Received: 28 February 2010 / Accepted: 8 October 2011 / Published online: 20 October 2011

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. In synchronous networks, too, the complexity of set agreement has been a significant research challenge that has now been resolved. Real systems, however, are neither purely synchronous nor purely asynchronous. Rather, they tend to alternate between periods of synchrony and periods of asynchrony. Nothing specific is known about the complexity of set agreement in such a "partially synchronous" setting.

In this paper, we address this challenge, presenting the first (asymptotically) tight bound on the complexity of set agreement in such systems. We introduce a novel technique for simulating, in a 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 simulation 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. Specifically, we show that every set agreement protocol requires at least $\lfloor \frac{t}{k} \rfloor + 2$ synchronous rounds to decide. We present an (asymptotically) matching algorithm that relies on a distributed asynchrony detection mechanism to decide as soon as possible during periods of synchrony. From these two results, we derive the size of the minimal window of synchrony needed to solve set agreement.

D. Alistarh (✉) · R. Guerraoui
Swiss Federal Institute of Technology, Lausanne, Switzerland
e-mail: dan.alistarh@epfl.ch

S. Gilbert
Department of Computer Science, National University of Singapore, Singapore, Singapore

C. Travers
LaBRI, University of Bordeaux 1, Talence, France

By relating synchronous, asynchronous and partially synchronous environments, 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 Gafni et al. (in SIAM J. Comput. 40(1):63–78, 2011), and to re-derive the combinatorial topology lower bound of Guerraoui et al. (in Theor. Comput. Sci. 410(6–7):570–580, 2009) in an algorithmic way.

Keywords Distributed computing · Message passing · Set agreement · Eventual synchrony · Time complexity · Lower bounds

1 Introduction

Set agreement was first introduced by Chaudhuri [8] to capture the power of allowing more choices than *consensus* [18], 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. The power of set agreement depends on the parameter k. When $k = 1$, set agreement reduces to consensus. When $k = n$, the problem is trivial, i.e., processes can act entirely independently.

Impossibility Results and Lower Bounds. In a collection of seminal papers, Borowski and Gafni [5], Herlihy and Shavit [16], and Saks and Zaharoglou [20] 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. [9] 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 $\lfloor \frac{t}{k} \rfloor + 1$ rounds are necessary. More recently, Gafni et al. [14] and Guerraoui et al. [15] 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 two different ways) that at least $\lfloor \frac{f}{k} \rfloor + 2$ rounds are needed.

Partial Synchrony. 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. (For example, consider a TCP network [6] under varying loads, which may affect the message delivery delays.) To describe such a system, Dwork et al. [11] 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 [9, 12] 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 then introducing an algorithm that terminates in any window of synchrony of size at least $\lfloor \frac{t}{k} \rfloor + 4$ rounds. Together, these results show that there exists an inherent price to tolerating asynchronous executions, and that this price is constant in the context of the set agreement problem.

Lower Bound by Reduction. The technique for deriving the lower bound is an important contribution, 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.

Our main tool is a technique for simulating "locally synchronous" executions in an asynchronous system. In particular, we show how to perform a k-fault-tolerant simulation of a *message-passing* system in an asynchronous *shared memory* system where each simulated execution *appears* synchronous to some processes.

This technique can be viewed as a generalization of the simulation technique of [12], moving from synchronous systems to cover the spectrum of partially synchronous ones. There are two new key observations. First, when the simulation is run for an epoch of length $\lfloor \frac{t}{k} \rfloor + 1$ rounds, we show that either some simulator sees a window of synchrony of size $\lfloor \frac{t}{k} \rfloor + 1$ rounds, or some simulator fails. Second, we observe that these epochs of length $\lfloor \frac{t}{k} \rfloor + 1$ can be repeated until either some simulator fails, or some simulator decides. From this we conclude that we have successfully simulated a set agreement protocol, resulting in the desired reduction.

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 [14, 15], using sophisticated techniques, that even in an execution with $f < t$ failures, some process cannot decide prior to round $\lfloor \frac{f}{k} \rfloor + 2$. Strictly speaking, these two results differ in how failures are counted. In [15], the lower bound is *global*: *some* process requires at least $\lfloor \frac{f}{k} \rfloor + 2$ rounds. In [14], the lower bound is *local*: *every* process decides after round $\lfloor \frac{f}{k} \rfloor + 2$. The latter bound applies in the case where the total number of processes n is unbounded and an unbounded number of failures can occur.

Using our simulation technique, we re-derive both lower bounds in a simpler and more general manner, in the standard model where t and n are bounded and known a priori. 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 $\lfloor \frac{f}{k} \rfloor + 1$ in executions with at most f failures, then in the worst-case, some process takes at least time $\lfloor \frac{t}{k} \rfloor + E(\cdot) + 1$ to decide (where E is a function of t, k and d).

Upper Bound for Eventually Synchronous Agreement. We then present the first known algorithm for k-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 $\lfloor\frac{t}{k}\rfloor + O(1)$. For simplicity, we show synchronous round complexity of $\lfloor\frac{t}{k}\rfloor + 4$. Closing the gap between these 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 $\lfloor\frac{t}{k}\rfloor + O(1)$ synchronous rounds. Even so, different sets of processes may have different views of the system when the decision occurs, since there are only $\lfloor\frac{t}{k}\rfloor + O(1)$ 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.

The key property of the algorithm is that there are at most k different high priority estimates in the system when a decision occurs. In a *synchronous* system, this would follow from the following fact: if there are $k + 1$ distinct estimates that remain at the end of a round, then there must have been at least k failures during that round. In a partially synchronous system, however, this is not true, as asynchronies can play the same role as failures in keeping extra values in the system. Instead, we rely on a careful analysis of the distributed asynchrony detection.

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 of computation, the *synchronous* model of computation, and the *asynchronous* model 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 Sect. 5 uses $t < n/2$.) The processes communicate via a message-passing network, modeled much as in [10, 11, 17]: time is divided into *rounds*. In each round, a process sends messages, receives messages, and performs some local computation. We assume that processes may fail by crashing. If a process p fails while sending messages in a round r, any subset of the messages that p sends in that round may be delivered to their recipients. A process that has not crashed by the end of round r is called *non-failed* at round r.

In this model, 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. (This can be implemented by delaying a round $r+1$ message until at least $n-t$ round r messages have been received.) 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. Similar round-based models have been studied by Charron-Bost and Schiper [7] (the heard-of model), by Keidar et al. [17] (the GIRAF model), and by Schmid et al. [4, 21] (the perception-based fault model).

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 $\Pi=\{p_1,p_2,\dots,p_n\}$, up to k of which may crash. The processes communicate via single-writer, multi-readers (SWMR) registers. The memory is organized in arrays $X[1\dots n]$ of n registers; entry $X[i]$ of an array can be written only by p_i. We assume that registers are initialized with an special value $\perp$. Also, for simplicity, we consider that each register is written at most once. (Note that our simulations have this property.)

In addition to read() and write() operations, a process can also invoke X.snapshot() to read all the contents of X in a logically instantaneous single operation. Let x and x' be the result of any two snapshot operations on X, possibly invoked at different processes. We assume that the following hold: *Containment:* $x\subseteq x'\vee x'\subseteq x$;[1] *Self inclusion:* Let v be the value written by p_i in $X[i]$ prior to invoking X.snapshot(), with no intervening $X[i]$.write($\cdot$) operations by any process; let x be the result of the snapshot operation; then $x[i]=v$. An implementation of snapshots on top of SWMR registers can be found in [1, 3]; thus, snapshots provide no extra computational power in this model. k-set agreement is known to be impossible in $AS_{n,k}$ [5, 16, 20].

Adopt-commit Objects. Our simulation relies on *adopt-commit* objects to coordinate which messages are delivered in each simulated round. An adopt-commit object AC, introduced in [12, 22], supports one operation propose(v) that returns a decision (dec, v) where $dec\in\{$adopt, commit$\}$. The object satisfies the following properties: *Termination:* Each invocation by a correct process terminates. *Validity:* If a process decides (dec, v) then some process invoked AC.propose(v). *Agreement:* If a process decides (commit, v), then every decision is $(\cdot, v)$. *Convergence:* If every process proposes the same v, then (commit, v) is the only possible decision.

Note that these properties ensure that the only case when distinct values v are returned by processes is when every process returns (adopt, $\cdot$). Wait-free implemen-

[1] For two vectors v and v', $v\subseteq v'$ if $\forall i, 1\le i\le n: v[i]\neq\perp\Rightarrow v[i]=v'[i]$. Recall that each register is written to at most once in our simulations.

tations of adopt-commit objects in $AS_{n,k}$can be found in [12, 22]. These implementations also satisfy: *Commit Validity:* Assume p_j invokes $AC.\mathsf{propose}(v)$; then p_j cannot get back (commit, v') with $v \neq v'$.

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

In this section, we present an algorithm for simulating executions of the partially synchronous model $ES_{n,t}$in the asynchronous system $AS_{n,k}$. Assuming an algorithm for solving k-set agreement in a window of synchrony of size at most $\lfloor \frac{t}{k} \rfloor + 1$, we use the simulation to derive a k-set agreement algorithm in $AS_{n,k}$.[2] This leads to a contradiction, as k-set agreement is impossible in the $AS_{n,k}$ model.

Preliminaries. Let $\mathcal{A}$ be a protocol designed for the round based model $ES_{n,t}$, and let α be an execution of $\mathcal{A}$. We can assume without loss of generality that algorithm $\mathcal{A}$ directs each non-failed process to send a message to all processes in each round.[3] A *trace* of the execution α is a sequence of vectors $(REC^1, REC^2, \ldots)$, with the property that vector REC^r is associated with round r and describes the set of messages received by each process in that round. In particular, if process p_i has not failed by the end of round r, $REC^r[i]$ is the set of processes from which process p_i receives messages in round r. If p_i has not failed by the end of round r, then we assume that it always receives its own message in that round, i.e., $p_i \in REC^r[i]$. On the other hand, if process p_i crashes during round r, then $REC^r[i] = \bot$. Also, since a failed (crashed) process does not recover, $REC^r[i] = \bot$ implies that $REC^{r+1}[i] = \bot$, and $p_i \notin REC^{r+1}[j]$ for each process p_j that has not failed by the end of round r. Moreover, by our model assumptions, any set $REC^r[j] \neq \bot$ is of size at least $n - t$.

A round r is *synchronous* if every non-failed process receives a message from each non-failed process in round r. A *window of synchrony of size* ℓ is a sequence of ℓ consecutive synchronous rounds. Formally, rounds $r_1, \ldots, r_1 + \ell - 1$ form a window of synchrony if the following properties hold: (1) $\forall r_1 \le r < r_1 + \ell - 1, \forall i, j$ such that $REC^{r+1}[j] \neq \bot$ and $REC^r[i] \neq \bot$, $REC^{r+1}[j] \subseteq REC^r[i]$ and (2) $\forall i, j$ such that $REC^r[j] \neq \bot$ and $REC^r[i] \neq \bot$, $p_j \in REC^r[i]$.

We say that process p_i has *a synchronous view of rounds* $r', r' + 1, \ldots, r$ in α if the state of p_i is the same at the end of round r and at the end of an execution α' that consists in r rounds and in which rounds $r', r' + 1, \ldots, r$ are synchronous.

Overview. The simulation pseudocode is presented in Fig. 1. The aim is to simulate an execution of algorithm $\mathcal{A}$ in model $ES_{n,t}$in which some processes have synchronous views of a large number of rounds, namely at least $\lfloor \frac{t}{k} \rfloor + 1$ consecutive rounds. The basic idea is similar to that of [12]—we simulate each synchronous round by writing messages to shared memory, and we then run a weak agreement protocol to determine which messages to "deliver" to each simulated process. In order to maintain synchronous views, we might have to *mute* some processes. Intuitively,

[2] It is essential for the parameter k of the set agreement problem to be the same as the maximum number of failures k among the simulators, since we reduce from the impossibility of k-set agreement in $AS_{n,k}$.

[3] Any algorithm $\mathcal{A}$ can be easily modified to satisfy this property.

```
Parameters:
Algorithm $\mathcal{A}$, number of phases $numP$, round array $[R_1, \ldots, R_{numP+1}]$
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. Each entry is initially $\bot$.
Local variables:
$S_i$, $sFlag_i$, variables with global scope
procedure propose($v_i$): start Task T1; start Task T2;
Task T1:
  $(\_, m_i) \leftarrow$ compute($0, v_i$, true) % messages for the first round
  for $\rho = 1$ to $numP$ do
      % Begin a new phase:
      $S_i \leftarrow \emptyset$ ; $sFlag_i \leftarrow$ true % these variables will be modified in simulate
      for $r = R_\rho$ to $R_{\rho+1} - 1$ do
          $rec_i \leftarrow$ simulate($m_i, r$) % Simulate send/receive of round $r$.
          $(d_i, m_i) \leftarrow$ compute($r, rec_i, sFlag_i$) % Compute message for the next round.
          if $d_i \neq \bot$ then $DEC[i]$.write($d_i$); stop T2; return $d_i$
Task T2:
  repeat for $j = 1$ to $n$ do $dec_i[j] \leftarrow DEC[i]$ until $(\exists \ell : dec_i[\ell] \neq \bot)$
  stop T1; return $dec_i[\ell]$
procedure simulate( $m_i, r$ ) % Simulate round $r$ where $p_i$ sends message $m_i$.
  $rec_i \leftarrow \emptyset$; $VAL[r][i]$.write($m_i$)
  repeat $view_i \leftarrow VAL[r]$.snapshot() until $|\{j : view_i[j] = \bot\}| \leq k$
  $M_i \leftarrow \{j : view_i[j] = \bot\}$ ;
  for $j = 1$ to $n$ do
      if $j \in S_i \cup M_i$ then $state_i[j] \leftarrow AC[r][j]$.propose(suspect)
      else $state_i[j] \leftarrow AC[r][j]$.propose(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 $rec_i \leftarrow \{\langle j, view_i[j] \rangle : view_i[j] \neq \bot\}$ ; $sFlag_i \leftarrow$ false
  if $\langle i, m_i \rangle \notin rec_i$ then $rec_i \leftarrow rec_i \cup \{\langle i, view_i[i] \rangle\}$ ; $sFlag_i \leftarrow$ false
return $rec_i$
```

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

a muted process continues receiving messages, but its messages are not received by other, non-muted processes. If the message of some process p_j is not received by some process p_i in round r (this implies in a synchronous execution that p_j fails in round r), then allowing in round $r + 1$ the message of p_j to be delivered to p_i causes the view of p_i to be no longer synchronous. As our goal is to maintain a synchronous view for at least one process, it might thus be required to mute some processes. Muted processes may however receive arbitrary messages, even from other muted processes. As long as no messages from muted processes are received by a non-muted process, the views of the non-muted processes remain synchronous.

As we will see, in each simulated round, the messages from at most k non-muted processes may be delivered to some but not all processes. Thus, the simulation mutes at most k new processes per round, where k is the number of processes that may crash in $AS_{n,k}$. In the following, we refer to processes in $AS_{n,k}$as *simulators*. As at most k

simulated processes may be muted in a simulated round, by the end of the simulation of the first $\lfloor\frac{t}{k}\rfloor$ simulated rounds, at most t simulated processes may have been muted. Therefore, in round $\lfloor\frac{t}{k}\rfloor$, a simulated process p_i that has a synchronous view of the first $\lfloor\frac{t}{k}\rfloor$ rounds may receive as few as $n-t$ messages (but not fewer) from distinct processes. More precisely, at most $k\lfloor\frac{t}{k}\rfloor$ processes are muted in the simulation of the first $\lfloor\frac{t}{k}\rfloor$ rounds and in addition, at most k messages from non-muted processes might be not delivered to p_i in round $\lfloor\frac{t}{k}\rfloor+1$.

We are able to extend this synchronous view by one more round, i.e., we show that in round $\lfloor t/k\rfloor+1$, at least one process p_j has a synchronous view of size $\lfloor t/k\rfloor+1$. Thus, assuming an algorithm where every process decides by the end of round $GST+\lfloor t/k\rfloor+1$, we conclude that simulated process p_j must decide. Each process is simulated by one simulator. If the simulator of p_j does not fail, it then can write this decision in shared memory thereby enabling every other simulator to decide. Otherwise, the simulator of p_j fails. In this case, we continue, repeating the simulation for another $\lfloor t/k\rfloor+1$ rounds, again resulting in either a process deciding or the failure of its simulator. Eventually, after $k+1$ repetitions (which we refer to as *phases*), we argue that some process decides and its associated simulator does not fail.

This simulation implies a lower bound on the round complexity of k-set agreement in $ES_{n,t}$. We assume, for the sake of contradiction, that there exists an algorithm $\mathcal{A}$ for $ES_{n,t}$in which, for every execution, every correct process decides by the end of round $GST+\lfloor t/k\rfloor+1$. We then show that our simulation of $\mathcal{A}$ solves k-set agreement in $AS_{n,k}$, which is impossible [5, 16, 20].

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 indicates the first round in the ith phase, with $R_1=1$. That is, each phase i consists in $R_{i+1}-R_i$ rounds.

For process p_i, the algorithm $\mathcal{A}$ is described by a function compute(r, *rec*, *sFlag*), where r is a round number and *rec* a set of messages received by p_i in round r. (The third parameter, *sFlag*, indicates whether the view of p_i of the rounds of the phase is so far synchronous, and is used primarily in Sect. 4.) The 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 $\perp$, if no decision has been reached. Without loss of generality, we assume that each process sends the same message to all processes, including itself.

3.2 Simulating Synchronous Rounds

Each process in $AS_{n,k}$simulates one process in $ES_{n,t}$. We will refer to the processes in $AS_{n,k}$as *simulators*. We denote sim_i the simulator in $AS_{n,k}$that simulates the process p_i in $ES_{n,t}$. The simulation begins with a call to propose(v_i) (line 8), where v_i is sim_i's proposal (recall that the aim of the simulators is to solve k-set agreement in $AS_{n,k}$). The simulation is divided into *phases* (lines 11–17); each phase is divided into *rounds*.

Simulation Overview. The simulation at sim_i begins with an invocation of propose(v_i) (line 8), where v_i is sim_i's proposal and also the input to $\mathcal{A}$ of the simulated process p_i. This launches two tasks $T1$ and $T2$ that run in parallel. In task $T1$ (lines 9–17), sim_i simulates steps of algorithm $\mathcal{A}$ in order to obtain a decision d_i for p_i. When a decision is reached, sim_i writes it in $DEC[i]$, decides, and exits (line 17). In task $T2$ (lines 18–20), sim_i periodically reads the shared array DEC. When it observes a non-$\bot$ value, the simulator decides that value and exits.

Round Overview. In order to simulate round r (lines 14–17), simulator sim_i invokes simulate(m_i, r) (line 15), where m_i is p_i's message for round r, which was computed at the end of the simulation of the previous round. The simulate procedure returns a set of pairs $\langle j, m_j \rangle$, where m_j is the message received from p_j by p_i in the simulated round r, and modifies the local variables S_i and $sFlag_i$. The simulator then calls the compute function (line 16), which returns d_i, a possible decision, and m_i, the next message to send.

Failed, Muted and Suspected Processes. A simulated process p_i *fails* in the simulated execution whenever its dedicated simulator sim_i fails. Let us fix a phase ρ. To simplify the discussion, the rounds of this phase are numbered $1, \ldots, R$. The goal is to simulate an execution in which in each phase a process has a synchronous view of the rounds of the phase.

To that end, each simulator maintains a set of *suspected* processes S_i and a flag $sFlag_i$. The set is emptied and the flag is set to true at the beginning of each phase (line 11). $sFlag_i =$ true at the end of round r indicates that process p_i has a synchronous view of rounds $1, \ldots, r$. The fact that process p_j is in S_i at the end of round r means that simulator sim_i suspects that there exists some round $r' \leq r$ in which the message of p_j was not delivered to every process. Suspicions might not be accurate but they are complete in the following sense: if the message of p_j is not delivered to some non-failed process in round r', then p_j is suspected by every simulator by the end of the *next* round $r' + 1$. Process p_j is muted at round r if it is suspected by every non-failed simulator at the end of round r. As within a phase no processes are ever removed from the sets S_i, a muted process never recovers from this state during a phase. Furthermore, we ensure that for every muted process p_j at round r, no process p_i with $sFlag_i =$ true delivers a message from p_j in round $r + 1$, for every round r of phase ρ. This property is central to show the existence of a process with a synchronous view at the end of each phase.

Simulating a Round. The simulate function (lines 21–34) carries out the send/receive step. For round r, simulator sim_i writes the message m_i into the register $VAL[r][i]$ (line 22), and then performs repeated snapshots of $VAL[r]$ (line 23) to discover the messages proposed by other simulators. Since k simulators may fail in $AS_{n,k}$, the simulator cannot wait for all n simulators to write a value to the array $VAL[r]$. As soon as sim_i discovers $(n - k)$ messages in its snapshot of $VAL[r]$, it continues. The variable M_i then stores the set of up to k processes from which a message has not been received in this simulated round. Since the array $VAL[r]$ is read by snapshot operations, the sets M_i are ordered by containment. Moreover, the largest set is of size at most k.

The simulators then agree on which messages to deliver in round r using a sequence of n adopt-commit objects (lines 25–30). Simulator sim_i records the set of messages p_i receives in round r in the local variable rec_i, which is empty at the beginning of the simulation of the round (line 22). If a simulator sim_i misses a message from a process p_j in round r (i.e., if $p_j \in M_i$), or if simulator sim_i suspects p_j (i.e., if $p_j \in S_i$), then it proposes suspecting p_j to the jth adopt-commit object $AC[r][j]$ of the sequence (line 26). Otherwise, the simulator proposes that p_j is *alive* (line 27). Three decisions are possible.

1. (commit, *suspect*) (line 28): In this case, the simulator mutes process p_j in round r. By *agreement*, we know that every simulator either adopts or commits to suspecting p_j, and so every non-failed simulator sim_ℓ adds p_j to S_ℓ. The round r message m_j of p_j (if any) is not received by p_i. This is materialized by the fact that rec_i remains unchanged.
2. (adopt, *suspect*) (line 29): In this case, we cannot determine whether p_j is simulated as muted or not in round r, as the decision of other simulators may be (adopt, *suspect*), (commit, *suspect*), or (adopt, *alive*); even so, to be safe, simulator sim_i adds p_j to S_i. We know, however, by *validity*, that some process proposed p_j as alive, and so $VAL[r][j]$ must contains the message from p_j, which we add to the set rec_i of messages to be received.
3. ($\cdot$, *alive*) (line 30): As in the second case, we add the message from $VAL[r][j]$ to rec_i.

Notice that if any simulator commits to *suspect* p_j, then by *agreement* every other simulator sim_ℓ either adopts or commits to *suspect* p_j and adds p_j to S_ℓ. Then, in the following round, every simulator proposes *suspect* p_j (line 26) which implies by *convergence* that every simulator commits to *suspect* p_j. It thus follows that the message from p_j, if any, is ignored. By using the adopt-commit objects in this way, we ensure that once a process is simulated as muted, it stays in this state in each subsequent round.

The End of the Phase. This approach results in not delivering messages from up to k new processes in each round (see Lemma 6). Eventually, the set of messages received by a process may fall below $n - t$, the bound on the minimal number of messages received per round in $ES_{n,t}$. In this case, not all simulated processes may maintain a synchronous view. We establish however the existence of a process that receives at least $n - t$ messages per round and has a synchronous view of size $\lfloor \frac{t}{k} \rfloor + 1$ at the end of the phase (Lemma 4).

If simulator sim_i discovers that the set of messages rec_i is too small or does not contain the message of p_i, the set rec_i is augmented to ensure that it contains enough messages ($|rec_i| \geq n - t$, line 32) and that it contains the round r message of p_i (line 33). This augmentation is always possible since the number of missing messages in the array $VAL[r]$ is bounded by $k \leq t$ and hence p_i observes at least $n - t$ round r messages. Since the view of p_i is then no longer synchronous, the flag $sFlag_i$ is set to false.

Finally, we examine whether and when processes decide. Assume we are simulating an execution of a set agreement protocol that decides by round $GST + \lfloor t/k \rfloor + 1$, and assume that each phase is of size at least $\lfloor t/k \rfloor + 1$. Then, since at least one simulated process p_i has a synchronous view of the entire phase, we conclude that p_i

decides by the end of the phase. Either the simulator of p_i fails, or it writes the decision to the shared memory $DEC[i]$. In the latter case, every other simulator eventually observes the decision (lines 18–20) and terminates. Thus, if no decision is reached, then a simulator fails in each phase. Since there are only k possible failures in $AS_{n,k}$, by the end of phase $k+1$ every simulator reaches a decision, completing a successful simulation of a k-set agreement protocol for $ES_{n,t}$in $AS_{n,k}$.

3.3 Analysis of the Simulation

We now provide some basic lemmas showing that the simulation is correct. The main claims are Lemma 2, which shows that the simulated execution is a correct execution of $ES_{n,t}$, and Lemma 4, which shows that in every phase, there is at least one process that has a synchronous view of the entire phase.

We say that a simulator *participates* in the simulation of round r if it reaches the r-th iteration of the inner loop of task T1 (line 15). When we refer to the value of the variable var_i of some simulator sim_i at some point in the execution, we implicitly assume that at this point sim_i has not failed. We first argue that the simulation is non-blocking. The only blocking statement is the **repeat** on line 23; since there are at most k failures, it never delays a simulator forever:

Lemma 1 *If no simulator decides and writes its value to DEC prior to round* r, *then no simulator is blocked forever while simulating round* r.

Proof The only possibility for a simulator to be blocked while simulating a round is in the **repeat** statement of line 23. Fix $r' \leq r$ to be the smallest round such that an invocation of $\mathsf{simulate}(\cdot, r')$ by a correct simulator p_i never terminates. As no simulator has decided while simulating rounds $1, \ldots, r'-1$, and there are at most k failures possible in the system, at least $n-k$ simulators eventually start simulating round r'. Therefore, the number of non-$\bot$ entries in $VAL[r']$ is eventually $\geq n-k$. Consequently, every participating simulator terminates the simulation of round r'. □

Next, we observe that the algorithm simulates an execution of $\mathcal{A}$ in $ES_{n,t}$, meaning that there is an execution of $ES_{n,t}$where each process sends and receives the same messages as in the simulation.

Lemma 2 *For every* $r \geq 1$, *there exists an execution* α *of* $ES_{n,t}$*executing* $\mathcal{A}$ *where in every round* $r' \leq r$ *of* α, *every process* $p_j \in \Pi$ *receives exactly the messages returned by* $\mathsf{simulate}(m_i, r')$.

Proof For contradiction, let $r \geq 1$ be the first round for which no such execution α exists. Let α be the $r-1$ round execution that satisfies the requirements of the lemma through round $r-1$, i.e., such that for every $p_j \in \Pi$, for every round $r' \leq r-1$, process p_j receives exactly the set of messages returned by $\mathsf{simulate}(m_j, r')$ in round r' of α.

Fix some process p_j that does not receive the messages returned by the call to $\mathsf{simulate}(m_j, r)$ in round r of α. First, it is easy to observe that every message returned by the call to simulate(m_j, r) was sent by some process in round r of α, as every such message was previously written in $VAL[r]$, and hence was computed (line 16) at the end of round $r-1$. Second, notice that the set *rec* returned is of size at least $n-t$: otherwise, additional messages are selected from $view_j$ to ensure that this is the case (line 32); moreover, it is clear that the simulate procedure only proceeds when $|view_i| \geq n-k$. Thus we can extend the execution α with the delivery to p_j of the messages returned by the call to simulate(m_j, r). Execution α remains a valid execution of $ES_{n,t}$, contradicting our hypothesis that no such execution existed. □

For each simulator p_i, let REC_i^r denote the value of the variable rec_i after p_i has executed the adopt-commit protocol, and *before the completion steps* of line 32 and line 33. That is, REC_i^r is the value of rec_i on line 31 of the instance simulate(m_i, r). We say that $p_j \in \text{REC}_i$ if $\langle j, \_\rangle \in rec_i$. Let S_i^r be the value of S_i when sim_i completes the simulation of round r. The set $S[r] = \bigcup_{sim_i \in \Pi} S_i^r$ is the set of suspected processes at the end of the simulation of round r. We now show that, within a phase, each REC_i^r set could have been received in a synchronous execution. In particular, if a process p_i does not receive a message in round r from some process p_ℓ ($p_\ell \notin \text{REC}_i^r$), then p_ℓ is simulated as muted in round $r+1$, and no sets $\text{REC}_j^{r'}$ with $r' > r$ ever again contain a message from p_ℓ. This follows from the agreement and convergence properties of adopt-commit objects.

Lemma 3 *For every round r in phase ρ (except for the last), for every $p_i, p_j \in \Pi$: $\text{REC}_j^{r+1} \subseteq \text{REC}_i^r$.*

Proof Fix p_i, p_ℓ and r such that for some round r, $p_\ell \notin \text{REC}_i^r$. Then we conclude that the status of p_ℓ from p_i's point of view is (commit, *suspect*). Due to the agreement property of adopt-commit, for every participating simulator sim_j, the state of p_ℓ is $(\cdot, \mathit{suspect})$, and so sim_j adds p_ℓ to S_j^r. Thus, in round $r+1$, every participating simulator proposes *suspect* for p_ℓ. Due to the convergence property of adopt-commit, every simulator gets back (commit, *suspect*) for p_ℓ, and hence no set REC_j^{r+1} includes p_ℓ. □

We now show that some process has a synchronous view of size $r - R_\rho + 1$ for every round $R_\rho \leq r \leq R_{\rho+1} - 1$. In other words, there exists an execution α of the system $ES_{n,t}$in which (1) some process p_i receives exactly the same sets of messages in α as in the simulation and, (2) every round $R_\rho, \ldots, r$ in α is synchronous.

Lemma 4 *Let r be some arbitrary round in phase ρ. If there is some simulator sim_i such that $p_i \in \text{REC}_i^r$ and $|\text{REC}_i^r| \geq n - f$, for some $f \leq t$, then there is an execution α of $ES_{n,t}$executing $\mathcal{A}$ such that* (1) *every round $R_\rho, \ldots, r$ is synchronous in α,* (2) *process p_i receives exactly the set of messages returned by* simulate$(_, r')$ *in each round r' of α and,* (3) *at most f processes fail in α.*

Proof By Lemma 2, there exists a $(R_\rho - 1)$-rounds execution β of system $ES_{n,t}$in which each process receives exactly the set of messages returned by the successive invocations of simulate$(_, r'')$ in each round $1 \leq r'' \leq R_\rho - 1$. Let γ be the suffix of

β defined as follows. Without loss of generality, we assume that in β no process has failed by the end of round $R_\rho - 1$. For every round r' in $\{R_\rho, \dots, r-1\}$:

1. $\forall p_j \in \Pi$, process p_j fails in round r' if and only if p_j has not failed prior to round r' and there exists a simulator sim_ℓ that does not simulate the reception of the message from p_j in round r', i.e., $p_j \notin \text{REC}_\ell^{r'}$.
2. For every pair of processes p_j, p_ℓ that have not failed by the end of round $r'-1$ according to the previous item, process p_ℓ receives a message from process p_j in round r' if and only if we have at simulator sim_ℓ $p_j \in \text{REC}_\ell^{r'}$.

In round $r' = r$, process $p_j \in \Pi$ fails if and only if p_j has not failed by the end of round $r-1$ and $p_j \notin \text{REC}_i^r$. In that case, no processes receive a message from p_j in round r.

Let $\alpha = \beta \cdot \gamma$. Consider a round r' in $\{R_\rho, \dots, r\}$. Let p_j and p_ℓ denote a pair of processes that have not failed by the end of round r'. It follows from item 1 above that $p_j \in \text{REC}_\ell^{r'}$ and therefore p_ℓ receives a message from p_j in round r'. Moreover, for every round $r' < r$ in phase ρ and every pair of processes p_j, p_ℓ that have not failed before round $r'+1$, $\text{REC}_i^{r'+1} \subseteq \text{REC}_j^{r'}$ by Lemma 3. Thus, γ forms a window of synchrony. Finally, by construction, at most f failures occur in execution α and process p_i receives the same sets of messages in α and in the simulation. □

Let M_i^r denote the set of simulator ids from which sim_i misses messages at line 24 in the invocation of $\mathsf{simulate}(\cdot, r)$. Let $view_i^r$ denote the value of the variable $view_i$ at simulator sim_i after the repeat loop (line 23). We next establish that for every round r, the sets M_i^r at different simulators are ordered by containment.

Lemma 5 *Let $i_1 \leq \dots \leq i_x$ the ids of the simulators that invoke* $\mathsf{simulate}(\cdot, r)$ *and execute line 24 in these instances. Denote X this set. There exists a bijection $\sigma : X \to \{1, \dots, |X|\}$ such that $M_{i_{\sigma(1)}}^r \subseteq \dots \subseteq M_{i_{\sigma(x)}}^r$. Moreover, we have $view_{i_{\sigma(1)}}^r \supseteq \dots \supseteq view_{i_{\sigma(x)}}^r$.*

Proof The array $VAL[r]$ is read by each simulator sim_{i_j} in snapshots. By the containment property of snapshot operations, the views $view_{i_j}^r$ obtained by each simulator at line 23 are ordered by containment. Let $\sigma : X \to \{1, \dots, |X|\}$ a bijection such that $\sigma(i_j) \leq \sigma(i_\ell)$ if and only if $view_{i_{\sigma(j)}} \supseteq view_{i_{\sigma(\ell)}}$, for every $i_j, i_\ell \in X$. It then follows that $M_{i_{\sigma(1)}}^r \subseteq \dots \subseteq M_{i_{\sigma(x)}}^r$ since $M_i^r = \{1, \dots, n\} \setminus \{\ell : view_i[\ell] = \bot\}$[4] for every $i \in X$. The second part of the lemma follows from the definition of σ. □

The next lemma shows a bound on the increase in suspicions in each simulated round:

Lemma 6 *For every r in phase ρ (except for the last), $|S[r+1] \setminus S[r]| \leq k$.*

[4] Recall that for two size n vectors v, v', $v \subseteq v'$ if and only if $\forall i, 1 \leq i \leq n : v[i] \neq \bot \Rightarrow v[i] = v'[i]$, given that our simulation has the property that a register is written to only once.

Proof During the simulation of round $r+1$, new suspicions may only be introduced when some simulator sim_i misses a round $r+1$ message at line line 23 from a process that has not been suspected before. Observe also that for every simulator sim_i, the set of missed messages has the property that $|M_i^r| \leq k$. Moreover, the sets M_i^r at different simulators are ordered by containment (Lemma 5). Consequently, at most k new suspicions are introduced in the simulation of round $r+1$, from which we conclude that $|S[r+1]\backslash S[r]| \leq k$. □

Finally, we show that new suspicions do not necessarily imply that less messages are received by all processes. Even when there are x new processes suspected in a simulated round, there are some processes that deliver the messages from these suspected processes. This fact allows us to extend the simulation one round further than it might be expected.

Lemma 7 *Let r be a round in phase ρ such that $|S[r-1]| \leq f$ for some $f \leq t$. Let $\Delta = S[r]\backslash S[r-1]$.* (1) *At least $n - |S[r]|$ simulators sim_i are such that $p_i \in \text{REC}_i^r$ and* $|\text{REC}_i^r| \geq n - |S[r]|$; (2) *For every $x \leq |\Delta|$, there exist x simulators $sim_i \in \Delta$ such that $p_i \in \text{REC}_i^r$ and* $|\text{REC}_i^r| \geq n - f - (x-1)$.

Proof Let j such that $p_j \notin S[r]$. By definition of $S[r]$, no simulators decide (*commit*, *suspect*) of (*adopt*, *suspect*) for p_j in round r. Hence, every simulator sim_i adds p_j to REC_i^r (line 30). This holds in particular for each simulator sim_i such that $i \in \Pi \setminus S[r]$ which proves (1).

For (2), let $\mathcal{C} = \Pi\backslash S[r-1]$ denote the set of processes that have not been suspected by the end of round $r-1$. Fix $x \leq |\Delta|$. Consider the simulation of round r. We order the simulators in Δ according to the size of their snapshot[5] of $VAL[r]$ at the end of the **repeat** loop (line 23), breaking ties using the order of ids. Since snapshots are related by containment, a simulator with higher rank has a larger snapshot, missing fewer messages from other simulators. Notice also that due to the self-inclusion property of the snapshot operations, a simulator always finds its own round r message included in the snapshot.

Consider the last x simulators $sim_{y_1}, \dots, sim_{y_x}$ in the order defined above, and fix some simulator sim_{y_ℓ} for $1 \leq \ell \leq x$. It follows from Lemma 5 that (1) $M_{y_x} \subseteq \cdots \subseteq M_{y_2} \subseteq M_{y_1}$ and, (2) $y_\ell \notin M_{y_\ell}$. Observe also that by definition of $S[r]$, each simulator sees a value in $VAL[r]$ for every process $p_i \in \Pi\backslash S[r]$. And the only ids in $\mathcal{C}$ that can be missed by sim_{y_ℓ} are the ids of the simulators that are ordered after it, i.e., $\mathcal{C} \cap M_{y_\ell} \subseteq \{y_{\ell+1}, \dots, y_x\}$.

Consider the proposals made by the simulator sim_{y_ℓ} for the adopt-commit objects (line 25). Notice that sim_{y_ℓ} proposes *suspect* only for processes in $S_{y_\ell} \cup M_{y_\ell}$. By definition of $\mathcal{C}$, $\mathcal{C} \cap (S_{y_\ell} \cup M_{y_\ell}) = \mathcal{C} \cap M_{y_\ell} \subseteq \{y_{\ell+1}, \dots, y_x\}$. Consequently, for each $i \in \mathcal{C}\backslash\{y_{\ell+1}, \dots, y_x\}$, sim_{y_ℓ} proposes (*alive*) to the adopt-commit object associated with p_i. Therefore, it follows from the commit validity property of adopt-commit objects that sim_{y_ℓ} cannot decide (commit, *suspect*) for each process in the set $\mathcal{C}\backslash\{y_{\ell+1}, \dots, y_x\}$, from which we obtain that $\mathcal{C}\backslash\{y_{\ell+1}, \dots, y_x\} \subseteq \text{REC}_{y_\ell}^r$. Hence,

[5] Here, the size of a snapshot in the number of non-$\perp$ entries in the vector *view*.

$|\text{REC}^r_{y_\ell}| \geq |\mathcal{C}| - (x - \ell) = |\Pi| - |S[r-1]| \geq n - f - (x - \ell)$ and $p_{y_\ell} \in \text{REC}^r_{y_\ell}$, as desired. □

3.4 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. Therefore, any k-set agreement protocol requires at least $\lfloor t/k \rfloor + 2$ synchronous rounds to decide.

In Sect. 3.3, we showed that the simulation is consistent with an execution of $ES_{n,t}$. The crux of the proof is to establish that at least one simulated process has a synchronous view of the rounds of each phase. 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 $\lfloor t/k \rfloor + 1$, either such a process decides by the end the phase, having seen the entire phase as synchronous, or its dedicated simulator fails.

Informally, Lemma 4 indicates that whenever REC^r_i is the set of messages delivered to p_i in round r, p_i has a synchronous view of the first $r - R_\rho + 1$ rounds of the phase and sees $f = n - |\text{REC}^r_i|$ failures. That is, at the end of round r, the simulated execution is indistinguishable for p_i from an execution in which rounds $R_\rho, \ldots, r$ are synchronous and no more than f failures occur. As at most t processes may fail in model $ES_{n,t}$, we want to show that there exists simulator sim_i such that $p_i \in \text{REC}^R_i$ and $|\text{REC}^R_i| \geq n - t$, where R is the last round of the phase. By the code, the sets of messages received by the simulated process p_i is then REC^R_i, and thus per Lemma 4, p_i has a synchronous view of the entire phase. The desired property is derived from Lemmas 6 and 7. From Lemma 6, we obtain an upper bound on the number of suspected processes at the end of round $R - 1$, namely at most t, as well as an upper bound, k, on the number of newly suspected processes in round R. Recall that each suspected processes p_j may not be included in sets REC^R_i. This, however, may not hold for every simulator: by part (2) of Lemma 7, we have that, $|\text{REC}^R_i| \geq n - t$ for at least one simulator sim_i. Moreover, for such a simulator, $p_i \in \text{REC}^R_i$.

Lemma 8 *For every phase ρ, if no simulators decide and write their decision to DEC prior to the end of phase ρ, then at least one simulator that begins phase ρ fails before beginning phase $\rho + 1$.*

Proof Assume for the sake of contradiction that no simulators that begin phase ρ fail prior to the end of phase ρ, and that no simulators decide by the end of phase ρ.

Let $R_\rho, \ldots, R_D$ be the $\lfloor \frac{t}{k} \rfloor + 1$ simulated rounds of phase ρ. First, we bound the number of suspected processes $|S[R_D - 1]|$ at the end of round $R_D - 1$: at the beginning of the phase, $S[R_\rho] = \emptyset$ since every simulator empties S_i at the beginning of the phase (line 13); per Lemma 6, each round introduces at most k new suspicions;

hence, $|S[R_D - 1]| \leq k\lfloor t/k \rfloor \leq t$. Consequently, the precondition of Lemma 7 is satisfied.

Let $\Delta = S[R_D] \setminus S[R_D - 1]$. If $\Delta = \emptyset$, there must exist a simulator sim_ℓ such that $p_\ell \in \text{REC}_\ell^{R_D}$ and $|\text{REC}_\ell^{R_D}| \geq n - |S[R_D]| \geq n - t$ per property (1) of Lemma 7. If $\Delta \neq \emptyset$, per Lemma 7(2), there must also exist a simulator sim_ℓ such that $|\text{REC}_\ell^{R_D}| \geq n - t$ and $p_\ell \in \text{REC}_\ell^{R_D}$.

Finally, by Lemma 4, process p_ℓ has observed a valid execution of algorithm $\mathcal{A}$ in system $ES_{n,t}$in which rounds $R_\rho, \ldots, R_D$ appear synchronous to p_ℓ. Therefore, since there are $\lfloor t/k \rfloor + 1$ rounds in phase ρ, and since algorithm $\mathcal{A}$ guarantees a decision in any window of synchrony of size $\lfloor t/k \rfloor + 1$, either process p_ℓ outputs a decision at the end of round R_D and its simulator writes it to DEC or the simulator of p_ℓ fails, leading to a contradiction. □

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 $ES_{n,t}$(Lemma 2), and *termination* follows from Lemma 8, which shows that if there is no decision, then at least one simulator fails in every phase; since there are only k failures in $AS_{n,k}$, by the end of phase $k+1$, some simulator must decide.

Lemma 9 *The algorithm in Fig.* 1 *simulating $\mathcal{A}$ solves k-set agreement in $AS_{n,k}$.*

Proof Termination: Eventually, every correct simulator decides: Assume for the sake of contradiction that some correct simulator never decides, which implies that no simulator ever writes out a decision to DEC. By Lemma 8, we know that in each phase ρ, some simulator must fail. Moreover, by Lemma 1, simulators continue to complete each phase. Thus, $k+1$ simulators fail by the end of phase $k+1$, contradicting the fact that most k simulators can fail.

Agreement: $|\{v \mid \exists i : DEC[i] = v \wedge v \neq \bot\}| \leq k$, *Validity:* $\forall i : DEC[i] \neq \bot \Rightarrow DEC[i] = v$, where v is the initial value of some process: From Lemma 2, we know that the sets of messages produced at each round by the simulation are the sets of messages received by the processes in some execution of $\mathcal{A}$ in system $ES_{n,t}$. Thus, agreement and validity follows immediately from the same properties of $\mathcal{A}$. □

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 The Complexity of Early Deciding Synchronous Set Agreement

We now show that the simulation presented in Sect. 3 can be used to derive lower bounds on the round complexity of early deciding synchronous k-set agreement. We say that a k-set agreement algorithm $\mathcal{A}$ is early deciding if in every execution in which at most f failures occur, processes decide by the end of some early round $R + 1 < \lfloor t/k \rfloor + 1$. We make this more precise as follows.

Let $\mathcal{A}$ denote a synchronous k-set agreement algorithm. As our purpose is to establish lower bounds on the round complexity of set agreement algorithms, we assume without loss of generality that in every execution of $\mathcal{A}$ every process that has not failed sends a message to every process, including itself, in every round.[6] Given an execution of an algorithm that satisfies this property, we say that a process p_i *sees at most f failures* by the end of round r if p_i receives at least $n - f$ messages in round r. That is, at the end of round r, process p_i cannot distinguish the current execution from an execution in which at most f failures occur.

Definition 1 Let d and R be positive integers, and let $\mathcal{A}$ be a k-set agreement algorithm in the $S_{n,t}$model. We say that $\mathcal{A}$ is in $ED(R, d)$ if in every run of $\mathcal{A}$, for every f such that $\lfloor \frac{f}{k} \rfloor \le R$, among the x processes that see at most f failures, at least $\min(x, d)$ of them decide by the end of round $R + 1$.

The main result of this section shows that every k-set agreement algorithm in $ED(R, d)$ pays a penalty for deciding early in terms of its worst-case running time.

4.1 Main Result and Corollaries

The following theorem demonstrates an inescapable tradeoff between the number d of processes that can decide early, the early decision round $R + 1$, and the worst-case decision round $R_D = \lfloor t/k \rfloor + 1 + E$ for deciding under any circumstances.

Theorem 2 *Let k, t, R be integers such that $0 < k \le t < n$ and $\lceil \frac{k}{d} \rceil < \lfloor \frac{t}{k} \rfloor - R$. Then, for any $d > 0$, the following hold*:

1. *If $d \ge k$, then there is no algorithm in $ED(R, d)$*;
2. *If $d < k$, then any algorithm in $ED(R, d)$ has a run in which some process decides at round $R_D = (\lfloor \frac{t}{k} \rfloor + 1 + E(d, k, t, R))$ or later, where $E(d, k, t, R) = \lfloor \frac{d(\lfloor \frac{t}{k} \rfloor - R - 1) - k + (t \bmod k)}{k-d} \rfloor$.*

We know that every k-set agreement algorithm tolerating t failures requires $\lfloor t/k \rfloor + 1$ rounds to decide in the worst-case [9, 12]. This theorem shows that achieving property $ED(R, d)$ implies sub-optimal worst case time complexity. We say that E is the price of deciding very early.

Before discussing the proof, we state two corollaries. The first shows a (global) lower bound on the number of rounds for every process to decide early. It follows from Theorem 2 where $d = n$:

Corollary 1 *Let k, t, R be integers such that $0 < k \le t < n$ and $1 < \lfloor \frac{t}{k} \rfloor - R$. Every k-set agreement algorithm in $S_{n,t}$has a run with f failures, for some f such that $\lfloor f/k \rfloor \le R$, in which some process decides after round $R + 1$.*

[6] If this property does not hold for algorithm $\mathcal{A}$, it is not hard to see that $\mathcal{A}$ can be modified to satisfy it while retaining the same round complexity.

Proof For the sake of contradiction, let $\mathcal{B}$ be a k-set agreement algorithm such that for every f, $\lfloor \frac{f}{k} \rfloor \leq R$, in every run with f failures, every process decides by the end of round $R+1$. Notice that in every run of $\mathcal{B}$, each process that observes f failures with $\lfloor \frac{f}{k} \rfloor \leq R$ must decide by the end of round $R+1$. Thus $\mathcal{B}$ is in $ED(R,d)$ for $d \geq k$, which implies that $\mathcal{B}$ does not exist. □

This bound is tight; matching algorithms can be found in [14, 19]. Herlihy et al. prove the same result in [15].[7] However, their proof is based on combinatorial algebraic topology, whereas we rely on algorithmic reduction.

The second corollary states a (local) lower bound on the number of rounds needed for even one process to decide early, and relies on Theorem 2 where $d = 1$:

Corollary 2 *Let k, t, R be integers such that $1 < k \leq t < n$, and $2k - 1 < \lfloor \frac{t}{k} \rfloor - R$. Every k-set agreement algorithm in $S_{n,t}$with worst-case round complexity $\lfloor t/k \rfloor + 1$ has a run with f failures, for f such that $\lfloor f/k \rfloor \leq R$, in which no process decides by the end of round $R+1$.*

Proof Suppose for contradiction that there exists an algorithm $\mathcal{B}$ with worst-case round complexity $\lfloor t/k \rfloor + 1$ such that in any run with f failures, at least one process decides by the end of round $R+1$. Then $\mathcal{B}$ is in $ED(R,1)$. Note that for $R \leq \lfloor \frac{t}{k} \rfloor - 2k$ and $d = 1$, $E(d,t,k,R) = \lfloor \frac{(\lfloor \frac{t}{k} \rfloor - R - 1) - k + (t \bmod k)}{k-1} \rfloor \geq 1$. Therefore, the worst-case complexity of any algorithm in $ED(R, 1)$ is at least $\lfloor t/k \rfloor + 2$ by Theorem 2: a contradiction. □

A complementary local early deciding lower bound is derived in [14], for systems with an unbounded number of processes, in which an unbounded number of failures can occur. The two results are incomparable, since the models considered are distinct. By contrast, our theorem holds in the standard model in which the number of processes n and the number of failures t are both bounded and known.

Thus Theorem 2 not only allows us to derive previous lower bounds on local and global early decision, but also unifies those results by considering the more general question of the worst-case round complexity, given d processes that decide early.

4.2 Overview of the Analysis

Fix parameters k, t, R and d, E matching the conditions of Theorem 2. In this section, we focus on the (interesting) case where $d \leq k+1$ and $E \geq 0$. For contradiction, assume that there exists an early deciding k-set agreement algorithm $\mathcal{A}$ in $ED(R,d)$ such that $d \geq k$ or that has worst case round complexity $R_D < \lfloor \frac{t}{k} \rfloor + 1 + E$. We show that this implies the existence of a k-set agreement algorithm in $AS_{n,k}$by simulating $\mathcal{A}$ using the algorithm described in Fig. 1 with only one phase of length $R_D = \lfloor t/k \rfloor + 1 + E$.

[7] Although, for technical reasons, the statements of the results are expressed differently, a careful analysis of the arguments reveals that the claims are equivalent.

Assume, without loss of generality, that the $\mathsf{compute}(\cdot, \cdot, sFlag)$ procedure for algorithm $\mathcal{A}$ always returns $(\bot, \top)$ if $sFlag = \mathsf{false}$. (In the simulated execution, a process never receives $\top$ as a message since a process "sending" $\top$, i.e., whose simulator writes $\top$ to VAL is muted—see Lemma 10.)

The crux of the proof lies in identifying simulated processes that observe a synchronous execution with (1) no more than f failures, where $\lfloor f/k \rfloor \leq R$, by the end of round $R+1$ or (2) no more than t failures by the end of round $R_D < \lfloor t/k \rfloor + 1 + E$. Such a process decides because $\mathcal{A}$ is in the class $ED(R, d)$ and R_D is the worst-case round complexity of $\mathcal{A}$. If there are at least $k+1$ such processes, the dedicated simulator of at least one them is correct. This simulator can write the decision in shared memory, enabling other simulators to decide.

By a careful analysis of the sets of messages delivered to the processes in the first $R+1$ rounds of the simulation, we identify a non-empty set D of processes that either see less than f failures at the end of round $R+1$, or whose simulator fails while simulating the first $R+1$ rounds (Lemma 11).

A simulator whose simulated process decides writes this decision to shared memory, allowing this value to be decided by other simulators (line 17). If this does not occur, the simulation of rounds $r > R+1$ proceeds with at most $k - |D|$ failures remaining among the simulators. Thus we can simulate "more" rounds (Lemma 12). We are then able to identify a set of $k+1-|D|$ processes that see at most t failures at the end of round R_D, from which we conclude that at least one correct simulator obtains a decision (Lemmas 13 and 14). Finally, as there exists a single run of $S_{n,t}$executing $\mathcal{A}$ in which each process receives the same set of messages (Lemma 15), we conclude that at most k proposed values are decided.

4.3 Proof of Theorem 2

We proceed by contradiction. We assume the existence of an algorithm $\mathcal{A}$ in $ED(R, d)$ that solves k-set agreement in $S_{n,t}$such that $d \geq k$ or with worst-case round complexity $R_D < \lfloor \frac{t}{k} \rfloor + E + 1$. Notice if $E < 0$, $\mathcal{A}$ trivially does not exist since every k-set agreement has a run in which at least one correct process has not decided by the end of round $\lfloor t/k \rfloor$ [9, 12]. So, in the following we suppose that $E(d, t, k, R) \geq 0$. Moreover, we consider without loss of generality that $1 \leq d \leq k+1$, as each algorithm in $ED(R, d)$ is also in $ED(R, d')$ for every $d' < d$.

Notation. We first fix some notation.

- Let $F[r]$ denote the set of processes p_i whose dedicated simulator sim_i has failed before starting the simulation of round $r+1$.
- Let $D[r]$ denote the set of processes p_i whose dedicated simulator has decided before starting the simulation of round $r+1$.
- As in Sect. 3.3, $S[r] = \bigcup_{p_i \in \{p_1, \ldots, p_n\}} S_i^r$ is the set of the suspected processes at the end of the simulation of round r.

  Similarly, REC_i^r is the value of the variable rec_i at simulator sim_i at line 31 of the instance $\mathsf{simulate}(m_i, r)$ (and before the completion steps of line 32 and line 33).
- Let $G[r] = \{p_i : p_i \in \mathrm{REC}_i^r \wedge |\mathrm{REC}_i^r| \geq n - t\}$. $G[r]$ is the set of processes p_i that have $sFlag_i = \mathsf{true}$ at the end of round r. For each process $p_i \in G[r]$, p_i receives

in round r of the simulated execution a message from each process $p_j \in \text{REC}_i^r$. None of the processes in $G[r]$ have been muted by the end of round r.

- R_D is the worst-case decision round for algorithm $\mathcal{A}$.

Notice that Lemmas 1, 3, 4, 5 , 6 and 7 refer to only one phase of the simulation. Therefore, they still hold in the context of this proof. First, we use Lemma 3 to obtain that if a process is not in $G[r]$ for round r, then none of the processes in $G[r+1]$ will receive its messages in the simulation of round $r+1$. In other words, $G[r]$ is the set of alive processes at the end of round r in the simulated execution.

Lemma 10 *For every round $r \in \{1, \ldots, R_D - 1\}$ and process $p_\ell \in \Pi$, if $p_\ell \notin G[r]$, for all processes $p_i \in G[r+1]$, $p_\ell \notin \text{REC}_i^{r+1}$.*

Proof Assume that $p_\ell \notin G[r]$. By definition of $G[r]$, $p_\ell \notin \text{REC}_\ell^r \vee |\text{REC}_\ell^r| < n - t$. Consider a process $p_i \in G[r+1]$. By Lemma 3, $\text{REC}_i^{r+1} \subseteq \text{REC}_\ell^r$. $|\text{REC}_i^{r+1}| \geq n - t$ since $p_i \in G[r+1]$. Therefore, $|\text{REC}_\ell^r| \geq n - t$ and thus $p_\ell \notin \text{REC}_i^{r+1}$. □

The following lemma is central to the proof. It states that at least $\min(k, d)$ simulators decide or fail before they start simulating round $R+2$.

As algorithm $\mathcal{A}$ is in the class $ED(R, d)$, a process that has a synchronous view of the first $R+1$ rounds in which no more than $f_m = kR + k - 1$ failures occur might decide. We notice that (1) "late" simulators, namely simulators associated with processes that are newly suspected in round $R+1$, see at most f_m failures (putting together Lemmas 7(2) and 6, which gives an upper bound on $|S[R]|$). Moreover, if at most $kR + k - 1$ suspicions are generated, i.e., (2) $|S[R+1]| \leq kR + k - 1$, each non suspected process observes at most f_m failures at the end of round $R+1$ (Lemma 7(1)). By combining (1) and (2), we are able to identify at least $\min(k, d)$ simulators that either decide or fail, which proves the lemma.

Lemma 11 $|D[R+1] \cup F[R+1]| \geq \min(k, d)$.

Proof Let $f_m = kR + (k-1)$. Let p_i be a process such that $p_i \in \text{REC}_i^{R+1}$ and $|\text{REC}_i^{R+1}| \geq n - f_m$, i.e., simulator sim_i simulates the reception of at least $n - f_m$ messages including a message from p_i in round $R+1$. We know by Lemma 4 that there is a an $(R+1)$-rounds execution α of system $S_{n,t}$ with $f \leq f_m$ failures in which, for every round r, p_i receives exactly messages rec_i^r. Therefore, p_i may decide by the end of round $R+1$ as algorithm $\mathcal{A}$ is in the class $ED(R, d)$. Let CD be the set of processes that may decide in this way by the end of round $R+1$.

Next, let $\Delta = S[R+1] \setminus S[R]$ be the set of processes that are newly suspected during the simulation of round $R+1$. Let p_j be a process in Δ. It follows from Lemma 7 that $p_j \in \text{REC}_j^{R+1}$ and $|\text{REC}_j^{R+1}| \geq n - (|S[R]| + |\Delta| - 1)$. By Lemma 6 and as no processes are initially suspected, $|S[R]| \leq kR$. Also, the same lemma implies that $|\Delta| \leq k$. We thus obtain $|\text{REC}_j^{R+1}| \geq n - ((k+1)R - 1) = n - f_m$. Hence, for every process in Δ, the simulated execution is indistinguishable from a $(R+1)$-rounds synchronous execution in which at most f_m failures occur. Thus, $\Delta \subseteq CD$.

Consider now some process $p_j \in \Pi \setminus S[R+1]$. By Lemma 7(1), $p_j \in \text{REC}_j^{R+1}$ and $|\text{REC}_j^{R+1}| \geq n - |S[R+1]| = n - (|S[R]| + |\Delta|)$. If $|\Delta| \leq k-1$, then $|\text{REC}_j^{R+1}| \geq n - (kR + k - 1) = n - f_m$ and thus $\Pi \setminus S[R+1] \subseteq CD$. Therefore, $\Delta \cup (\Pi \setminus S[R+1]) \subseteq CD$, from which we obtain that $|CD| \geq |\Delta \cup (\Pi \setminus S[R+1])| = |\Pi \setminus S[R]| \geq n - kR \geq n - k(\lfloor \frac{t}{k} \rfloor - 1) \geq k + 1 \geq d$, as $n - t \geq 1$. Otherwise, we have $|\Delta| = k$ and thus, as $\Delta \subseteq CD$, $|CD| \geq k$. Therefore in both cases we have $|CD| \geq \min(k, d)$.

As algorithm $\mathcal{A}$ is in the class $ED(R, d)$, at least $\min(|CD|, d) \geq \min(k, d)$ processes in CD decide by the end of round $R+1$. A simulator whose simulated process decides departs from the simulation and decides (line 17), unless it fails. Hence, we have $|F[R+1] \cup D[R+1]| \geq \min(d, k)$. □

We now analyze suspicions generated in the simulations of the rounds $r \geq R+3$.

Lemma 12 *For every $r \geq R+3$, $|S[r] \setminus S[r-1]| \leq k - \min(k, d)$.*

Proof Let us first observe that a simulator sim_i that has decided or failed before starting to simulate round r does not participate in the simulation of round r. More precisely, for every $r' \geq r$, sim_i never writes in $VAL[r'][i]$ and consequently i in is the set $M_j^{r'}$ for each simulator sim_j that participates in round r'. So, in round r', every simulator proposes (*suspect*) for p_i, and, by the convergence property of adopt-commit objects, every simulator decides (*commit*, *suspect*) for p_i. Hence $p_i \in S[r']$. In particular, this means that for $r \geq R+2$, $D[R+1] \cup F[R+1] \subseteq S[r]$ (P1).

Let $r \geq R+3$ and $p_\ell \in S[r] \setminus S[r-1]$. This can occur only if a simulator sim_j misses the round r message of p_ℓ, i.e., $\ell \in M_j^r$. Per Lemma 5, we know that sets M_j^r are ordered by containment. Let M be the largest set. It thus follows that $S[r] \setminus S[r-1] \subseteq M$. Notice also that, by line 23 of the code, we have that $|M| \leq k$.

As noted earlier, for each simulator sim_i that does not participate in round r, we have $i \in M_j^r$ where sim_j is any participating simulator. In particular, this implies that $D[R+1] \cup F[R+1] \subseteq M$. Finally, note that $(D[R+1] \cup F[R+1]) \cap (S[r] \setminus S[r-1]) = \emptyset$ since $D[R+1] \cup F[R+1] \subseteq S[r-1]$ by property P1 noted above. Therefore, $|S[r] \setminus S[r-1]| \leq |M| - |D[R+1] \cup F[R+1]| = k - |D[R+1] \cup F[R+1]| \leq k - \min(k, d)$. The last inequality follows from Lemma 11. □

Next, we establish an upper bound of t on the number of suspected processes at the end of round R_D. If in round r the message from some process p_j is not delivered to some other process by the end of round r, p_j must be suspected, i.e., $p_j \in S[r]$. This upper bound is then used to prove that in the last round, some simulated processes receive at least $n - t$ messages.

Lemma 13 *$|S[R_D - 1]| \leq t - (k - \min(k, d))$ and $|S[R_D]| \leq t$.*

Proof It follows from the fact that $S[0] = \emptyset$ and from Lemma 6 that $S[R+2] \leq (R+2)k$. For the remaining round $r = R+3, \ldots, R_D - 1$, we know by Lemma 12 that $|S[r] \setminus S[r-1]| \leq k - \min(k, d)$. Hence,

$$|S[R_D - 1]| \leq k(R+2) + (R_D - (R+3))(k - \min(k, d))$$

We consider two cases, according to the value $\min(k, d)$.

- $k \le d$. In this case $\min(k, d) = k$, and thus $|S[R_D - 1]| \le k(R + 2)$. Moreover, R, k and d are such that $\lceil \frac{k}{d} \rceil < \lfloor \frac{t}{k} \rfloor - R$. Hence, $(R + 2) \le \lfloor \frac{t}{k} \rfloor$ from which we conclude that $|S[R_D] - 1| \le k(R + 2) \le t$.
- $k > d$. In that case we have:

$$\begin{aligned}|S[R_D - 1]| &\le k(R+2) + (R_D - (R+3))(k-d)\\ &\le k(R+2) - k(R+2) + d(R+2) + (R_D - 1)(k-d)\\ &\le d(R+2) + \left(\left\lfloor \frac{t}{k} \right\rfloor + E\right)(k-d)\\ &\le -d\left(\left\lfloor \frac{t}{k} \right\rfloor - R - 2\right) + t - (t \bmod k) + E(k-d)\end{aligned}$$

By definition of E, $E(k - d) \le d(\lfloor \frac{t}{k} \rfloor - R - 1) - k + (t \bmod k)$. Therefore,

$$|S[R_D - 1]| \le d - k + t = t - (k - \min(d, k))$$

As observed earlier, at most $k - \min(k, d)$ new suspicions are generated in round R_D (Lemma 12). Therefore, $|S[R_D]| \le |S[R_D - 1]| + k - \min(k, d)$ and thus $|S[R_D]| \le t$. □

We can now state the main termination Lemma. It follows from Lemma 11 and from the analysis of the number of suspicions generated during the simulation of rounds $R + 3, \dots, R_D$ (Lemmas 12 and 13).

Lemma 14 (*Termination*) *Every correct simulator decides.*

Proof Let us assume for contradiction that some correct simulator never decides. As every correct simulator eventually decides if one simulator writes a decision value in the shared array *DEC* (Task T2), it follows that no simulator ever writes a decision value to *DEC*. Consequently, for all rounds r, we have that $D[r] = \emptyset$. It follows from Lemma 1 that every correct simulator completes the simulation of rounds $1, \dots, R_D$, for a total number of at most $\lfloor \frac{t}{k} \rfloor + E + 1$ simulated rounds. We know by Lemma 11 that at least $\min(d, k)$ simulators must have failed during the simulation of the first $R + 1$ rounds. If each of these simulators fails before deciding, we show that one simulated process has a synchronous view of the R_D rounds in which at most t failures occur. Moreover, the dedicated simulator of this process is correct. Hence, this simulator must decide: a contradiction.

Consider the last round R_D. Let $\Delta = S[R_D] \setminus S[R_D - 1]$ and $X = \Delta \cup (\Pi \setminus S[R_D])$. For every $p_i \in X$, it follows from Lemma 7 that $p_i \in \mathrm{REC}_i^{R_D}$ and $|\mathrm{REC}_i^{R_D}| \ge n - |S[R_D]|$. Notice that $|S[R_D]| \le t$ (Lemma 13). So, by Lemma 4, each process $p_i \in X$ has a synchronous view of rounds $1, \dots, R_D$ in which at most t failures occur. Therefore, as in every execution of $\mathcal{A}$, every non-faulty process has decided by the end of round of R_D, each $p_i \in X$ must decide in round R_D. It remains to show that the dedicated simulator of at least one of the processes in X is correct.

Observe that $X = \Pi \setminus S[R_D - 1]$. Hence, $|X| = n - |S[R_D - 1]| \ge n - t + k - \min(k, d) \ge 1 + k - \min(k, d)$ by Lemma 13 and the fact that $n > t$. We then notice that by Lemma 11 and the assumption that no simulator decides, $|F[R + 1]| \ge$

$\min(k,d)$. Recall that $F[R+1]$ is the set of simulators that fail before starting the simulation of round $R+2$. As in total at most k simulators may fail, this means that among the simulators that participate in the simulation of rounds $R+2,\ldots,R_D$, at most $k-\min(k,d)$ are not correct. As $|X| \geq 1+k-\min(k,d)$, there exists $p_i \in X$ such that the associated simulator sim_i is a correct simulator. □

The next lemma establishes that the simulated execution is a valid execution of algorithm $\mathcal{A}$ executing in $S_{n,t}$.

Lemma 15 *Let r such that $G[r] \neq \emptyset$ and $|S[r]| \leq t$. There is an execution α of system $S_{n,t}$executing algorithm $\mathcal{A}$ such that $\forall r' \leq r, \forall p_i \in G[r']$, process p_i receives exactly the set of messages returned by* $\mathsf{simulate}(\_, r')$ *in round r' of α.*

Proof Let α be the r-rounds execution defined as follows. $\forall r' \in \{1,\ldots,r\}$:

1. $\forall p_i \in \{p_1,\ldots,p_n\}$, process p_i fails in round r' if and only if p_i has not failed prior to round r' and $p_i \notin \bigcap_{p_j \in G[r']} \mathrm{REC}_j^{r'}$.
2. $\forall p_i \in G[r']$, process p_i receives a message from each process $p_j \in \mathrm{REC}_i^{r'}$ during round r'. $\forall p_i \notin G[r']$, process p_i receives a message from each process in $\bigcap_{p_j \in G[r']} \mathrm{REC}_j^{r'}$.

We first verify that α is a valid synchronous execution. Let $r' \in \{1,\ldots,r\}$. Let p_i denote a process that has not failed by the end of round r'. By definition, $p_i \in \bigcap_{p_j \in G[r']} \mathrm{REC}_j^{r'}$ and it thus follows from the second condition that every non-failed process receives a message from p_i in round r'. Suppose now that some process does not receive process p_i's message in round r'. Hence, $p_i \notin \bigcap_{p_j \in G[r']} \mathrm{REC}_j^{r'}$, and there exists a process $p_\ell \in G[r']$ such that $p_i \notin \mathrm{REC}_\ell^{r'}$. It then follows from Lemma 3 that every p_j has the property that $p_i \notin \mathrm{REC}_j^{r'+1}$. In particular, no processes in $G[r'+1]$ receive p_i's message during round $r'+1$, and no processes in $\Pi - G[r'+1]$ receive p_i's message since $p_i \notin \bigcap_{p_j \in G[r'+1]} \mathrm{REC}_j^{r'+1}$.

Second, we count the number of failures in α. Process p_i fails if and only if $p_i \notin \bigcap_{p_j \in G[r']} \mathrm{REC}_j^{r'}$ for some $r' \leq r$. As per Lemma 10, $G[r'+1] \subseteq G[r']$, $\Pi - \bigcap_{p_j \in G[r]} \mathrm{REC}_j^{r}$ is the set of faulty processes in α. Thus, p_i fails implies that $p_i \notin \mathrm{REC}_j^{r}$ for some processes p_j in $G[r]$. Hence, by definition of $S[r]$, $p_i \in S[r]$. Therefore, at most t processes fail in execution α, since we assume that $|S[r]| \leq t$.

Finally, by the definition of $G[r']$, $\forall p_i \in G[r']$, $\mathsf{simulate}(\_, r')$ returns $\mathrm{REC}_i^{r'}$. Therefore, it follows from the definition of α that every process in $G[r']$ receives the same sets of messages in the simulation of round r' and in the r'th round of execution α. □

Together with Lemma 14, the following lemma establishes that the simulators solve k-set agreement in model $AS_{n,k}$by simulating an execution of $\mathcal{A}$ in model $S_{n,t}$.

Lemma 16 (Agreement) $|\{v \mid \exists i : DEC[i] = v \wedge v \neq \bot\}| \leq k$.

(Validity) $\forall i : DEC[i] \neq \bot \Rightarrow DEC[i] = v$ *where v is the initial value of some process.*

Proof Every decision value v written in the array *DEC* is computed during the simulation of some round $r \leq R_D$ and v is the decision of some process p_i in the simulated execution. When simulator sim_i decides at round r, the simulated process p_i has not failed in the simulated run, i.e, $p_i \in G[r]$.

Let $r' \leq R_D$ the last round in the simulated execution in which a decision occurs. $S[r'] \subseteq S[R_D]$ and thus $|S[R_D]| \leq t$ by Lemma 13. Therefore, it follows from Lemma 15 that there exists a r'-rounds execution α of system $S_{n,t}$executing $\mathcal{A}$ such that, for every $r, 1 \leq r \leq r'$ and every $p_i \in G[r]$, $\mathsf{simulate}(\_, r)$ returns the set of messages received by p_i in round r of α.

The values written out in *DEC* are a subset of the values decided in α. The correctness of $\mathcal{A}$ thus implies that decision values written in *DEC* satisfy validity and agreement. □

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

In this section, we present an algorithm named K4 which solves k-set agreement in a window of synchrony of size $\lfloor t/k \rfloor + 4$. This is the first algorithm, to the best of our knowledge, for k-set agreement in $ES_{n,t}$. The pseudocode can be found in Fig. 2.

5.1 Description

K4 is a round-based full-information protocol, and it assumes that t, the number of failures, is less than $n/2$. 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 11–13), or if it sees $\lfloor t/k \rfloor + 4$ consecutive synchronous rounds (line 16). In case it decides, the algorithm returns the decided value est_i to the caller (lines 14 and 17), and continues to run the protocol by sending messages announcing its decision. If no decision is reached, then the estimate est_i is updated in lines 19–22. 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. The procedure $\mathsf{updateSynchDetector}()$ merges information into the *Active* and *Failed* sets; if a process believes that p_ℓ was active in round r (e.g., it receives a message from p_ℓ), then p_ℓ is added to $Active[r]$; if it believes that p_ℓ was failed during round r (e.g., it did not receive a message from p_ℓ), then p_ℓ is added to $Failed[r]$ (see lines 25–28). It then determines based on $Active[r]$ and $Failed[r]$ sets whether round r seems synchronous (lines 29–33). 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 $k > r$

```
procedure propose($v_i$)$_i$
    $est_i \leftarrow v_i$; $r_i \leftarrow 1$; $msgSet_i \leftarrow \emptyset$; $sFlag_i \leftarrow$ false
    $Active_i \leftarrow [\,]$; $Failed_i \leftarrow [\,]$; $AsynchRound_i \leftarrow [\ ]$
    while true do
        send( $est_i$, $r_i$, $sFlag_i$, $Active_i$, $Failed_i$, $AsynchRound_i$, $decided_i$ ) to all
        wait until received messages for round $r_i$ from at least $n - t$ processes
        $msgSet_i[r_i] \leftarrow$ messages that $p_i$ receives in round $r_i$
        $Active_i[r_i] \leftarrow$ processes from which $p_i$ gets messages in round $r_i$
        $Failed_i[r_i] \leftarrow \Pi \setminus Active_i[r_i]$
        updateSynchDetector() % Update the state of $p_i$ based on messages received.
        % Has anyone else decided?
        if ( $\exists$ process $p$ such that $msg_p \in msgSet_i$ with $msg_p.decided_p =$ true ) then
            $decided_i \leftarrow$ true
            $est_i \leftarrow msg_p.est_p$
            return $est_i$
        if ($sCount_i = \lfloor t/k \rfloor + 4$) then
            $decided_i \leftarrow$ true
            return $est_i$
        if ($decided_i =$ false) then
            % Identify the processes $p$ that have $sFlag_p$ set at the previous round
            $flagProcs_i \leftarrow \{\ p \in Active_i[r_i] \mid sFlag_p[r_i - 1] =$ true $\}$
            if $flagProcs_i \neq \emptyset$ then
                $est_i \leftarrow \min_{q \in flagProcs_i}(est_q)$ % Adopt minimum flagged estimate.
            else $est_i \leftarrow \min_{q \in Active_i[r_i]}(est_q)$ % Otherwise, adopt minimum estimate.
        % increment round counter
        $r_i \leftarrow r_i + 1$
procedure updateSynchDetector()
    % Update the Active and Failed sets for each previous round based on messages
      received
    for every $msg_j \in msgSet_i[r_i]$ do
        for round $r$ from 1 to $r_i - 1$ do
            $Active_i[r] \leftarrow msg_j.Active_j[r] \cup Active_i[r]$
            $Failed_i[r] \leftarrow msg_j.Failed_j[r] \cup Failed_i[r]$
    % Analyze the current view to detect asynchrony
    for round $r$ from 1 to $r_i - 1$ do
        $AsynchRound_i[r] \leftarrow$ false
        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
    % The current round is assumed to be synchronous
    $AsynchRound_i[r_i] \leftarrow$ false
    % Compute the number of consecutive synchronous rounds seen
    $sFlag_i \leftarrow$ false
    $sCount_i \leftarrow \mathbf{max}_\ell(\forall r' \in [r_i - \ell, r_i], AsynchRound_i[r'] =$ false)
    % If the last $\lfloor t/k \rfloor + 3$ are seen as synchronous, then set $sFlag_i$
    if $sCount_i \geq \lfloor t/k \rfloor + 3$ then $sFlag_i \leftarrow$ true
```

Fig. 2 The K4 algorithm, at process p_i

(i.e., $p_\ell \in \mathit{Active}[k]$). Finally, process p_i sets a flag *sFlag* to true if it sees the previous $\lfloor t/k \rfloor + 3$ rounds as synchronous (line 37). Note that the sets $\mathit{Active}_i[r]$ and $\mathit{Failed}_i[r]$ need not be disjoint: this can occur in rounds where process p receives a message from a process q, but another process does not receive the message from q, either because of q's failure or because of asynchrony.

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 is awarded high priority; all other estimates are awarded normal priority. Process p_i stores prioritized estimates in $\mathit{flagProcs}_i$ (line 19), and adopts the minimum prioritized estimate, if one exists (line 21). Otherwise, process p_i adopts the minimum estimate received in the current round (line 22).

5.2 Analysis

We prove that the algorithm K4 solves k-set agreement in $ES_{n,t}$. *Validity* and *Termination* are straightforward, so we focus on showing *Agreement*. The proof of agreement is based on the idea that in order for processes to maintain at least $k+1$ distinct estimates, at least k failures have to occur in each round. This is obvious if the system is synchronous—and yet quite non-trivial when the system may be asynchronous for certain periods. We identify a trade-off between the number of processes that have a synchronous view of an execution suffix, and the number of distinct estimates that these processes can carry. In particular, we prove that processes which have a synchronous view of $\lfloor t/k \rfloor + 3$ consecutive rounds may hold at most k distinct estimates, which, after some consideration, implies that there is no execution of the algorithm in which processes decide on more than k values. The key lemma, whose proof is presented in full in Sect. 5.3, is the following.

Lemma 17 (Elimination) *Let $r_m > 0$ be a round and $p_1, p_2, \ldots, p_{k+1}$ be $k+1$ processes that, at the end of round r_m, perceive the previous $\lfloor t/k \rfloor + 3$ rounds as synchronous. Then at least two such processes have the same estimate.*

Assuming that Lemma 17 holds, we can prove that the K4 algorithm preserves agreement.

Theorem 3 (Agreement) *In every execution, processes decide on a set of at most k distinct values.*

Proof Consider an arbitrary execution of K4 and let r_d be the first round in which a process decides. Let p_d be a process that decides in round r_d and let Supp_d be the set of processes with $\mathit{sFlag} = \mathsf{true}$ at the beginning of round r_d—we say that processes in Supp_d *support decision* in round r_d.

First, notice that, since process p_d is the first process to decide, it must necessarily set the $\mathit{decided}_i$ variable to true on line 16 of the propose procedure. This implies that $\mathit{sCount}_i = \lfloor t/k \rfloor + 4$. Therefore, each of the processes whose message p_d receives in round r_d must have perceived the previous $\lfloor t/k \rfloor + 3$ as synchronous at the end of round $r_d - 1$ (otherwise, p_d also notices an asynchrony at line 33 of updateSynchDetector and cannot decide). Hence, each such process $s \in Supp_d$ must have $sFlag_s = \mathsf{true}$ at the end of round $r_d - 1$. Since p_d receives $n - t$ messages in every round, we obtain that $|Supp_d| \geq n - t$.

On the other hand, Lemma 17 ensures that processes in $Supp_d$ have at most k distinct estimates at the beginning of round r_d. Denote the set of these values by $\mathcal{V}_k$. We prove that decisions in round r_d or in later rounds are necessarily made on a value in $\mathcal{V}_k$.

First, if a process decides at the end of round r_d (in line 14), then it maintains its previous estimate. The deciding process supported decision at the end of the previous round, therefore its estimate is in $\mathcal{V}_k$. Second, if the process does not decide at the end of r_d, then, since $|Supp_d| \geq n-t > \lfloor n/2 \rfloor$, the process necessarily receives a message from a process in $Supp_d$ in round r_d. In this case, the process updates its estimate in line 21 of procedure propose(), thereby adopting the minimum estimate that it receives from a process in $Supp_d$, which is in $\mathcal{V}_k$. This implies that any decisions made in later rounds also occur on elements of $\mathcal{V}_k$.

Since $|\mathcal{V}_k| \leq k$, we conclude that all decisions in this execution occur on at most k distinct values, which is equivalent to k-agreement. □

5.3 Proof of the Elimination Lemma

In this section, we prove the following result:

Lemma 18 (Elimination) *Let $r_m > 0$ be a round and $p_1, p_2, \ldots, p_{k+1}$ be $k+1$ processes that, at the end of round r_m, perceive the previous $\lfloor t/k \rfloor + 3$ rounds as synchronous. Then at least two such processes have the same estimate.*

Notation. We proceed by contradiction. Suppose there exists a round $r_m > 0$ and processes $p_1, p_2, \ldots, p_{k+1}$ such that $est_1 < est_2 < \cdots < est_{k+1}$ and all these processes see the previous $\lfloor t/k \rfloor + 3$ rounds as synchronous at the end of round r_m (i.e., the processes have $sFlag = \mathsf{true}$ at the end of round r_m). For clarity, let $r_m = r_0 + \lfloor t/k \rfloor + 3$, with $r_0 \geq 0$ and define the set $\mathcal{P} = \{p_1, p_2, \ldots, p_{k+1}\}$. Also, we use the notation $[1, \ell]$ for the set $\{1, 2, \ldots, \ell\}$. In the following, we use a superscript to denote the round from which a local variable is perceived. For example, $Active_i^{r_0+2}[r_0+1]$ is the *Active* set of process p_i for round r_0+1, as seen from the end of round $r_0 + 2$. Unless otherwise stated, we omit the superscript when we consider variables from the end of round $r_0 + \lfloor t/k \rfloor + 3$.

Proof Outline. We begin by establishing some basic properties of the processes' views, in Propositions 1–4. We then aim to show that, in order for $k+1$ estimates to be maintaned in the system by the processes in $\mathcal{P}$ (i.e., the processes have $sFlag = \mathsf{true}$ at the end of round r_m), at least k failures have to occur in each round seen as synchronous by the processes in $\mathcal{P}$. Proposition 5 makes this intuition precise. Next, Proposition 6 establishes that there exists a process q from which all processes in $\mathcal{P}$ received a message in round $r_0 + \lfloor t/k \rfloor + 2$, and Proposition 7 shows that this process has to perceive k distinct failures per round in rounds $r_0 + 1, \ldots, r_0 + \lfloor t/k \rfloor$.

This implies that process q has already experienced at least $t - (t \bmod k)$ failures by the end of round $r_0 + \lfloor t/k \rfloor + 1$. Since each process in $\mathcal{P}$ receives a message from q, this implies that each of these processes may see at most $k - 1$ new failures in rounds $r_0 + \lfloor t/k \rfloor + 2$ and $r_0 + \lfloor t/k \rfloor + 3$. To conclude, we show that this number of failures is not enough to maintain $k + 1$ distinct values in the system through the end of round $r_0 + \lfloor t/k \rfloor + 3$, which contradicts our initial assumption on the existence of processes $p_1, \ldots, p_{k+1}$. We note that throughout the proof we assume the existence of the $k + 1$ processes in $\mathcal{P}$.

We first analyze the synchronous views of processes $p_1, p_2, \ldots, p_{k+1}$.

Proposition 1 (Synchrony) *Let r be a round and p be a process such that at the end of round r, p sees the previous $\ell > 0$ rounds as synchronous, i.e. $AsynchRound^r[r'] = \mathsf{false}, \forall r' \in [r - \ell + 1, r]$. Then*

$$Active^r_p[r - \ell + 1] \supseteq Active^r_p[r - \ell + 2] \supseteq \cdots \supseteq Active^r_p[r].$$

Proof Assume there exists a round $r - \ell + 1 \leq r' \leq r - 1$ such that $Active^r_p[r' + 1] \not\subseteq Active^r_p[r']$. Then there exists a process $q \in Active^r_p[r' + 1] \setminus Active^r_p[r']$, which implies that $q \notin Active^r_p[r']$, therefore, by the way the sets *Active* and *Failed* are built, $q \in Failed^r_p[r']$. It follows that $q \in Failed^r_p[r'] \cap Active^r_p[r' + 1]$, therefore r' is asynchronous from the point of view of p at round r : contradiction. □

The second proposition formalizes the intuition that if process p receives process q's message sent in some round r (either directly or through a relay), then p has the entire information about q's state at the end of round $r - 1$.

Proposition 2 (Information Gathering) *Let p_i and p_j be two processes in Π and let $r_c \geq r \geq 2$ be two rounds. If $p_i \in Active^{r_c}_j[r]$, then for any round $r' < r$, $Active^{r-1}_i[r'] \subseteq Active^{r_c}_j[r']$ and $Failed^{r-1}_i[r'] \subseteq Failed^{r_c}_j[r']$.*

Proof Fix $r > 0$ and $r' < r$. We proceed by induction on $r_c \geq r$.

Base case: If $r = r_c$, then p_j has received p_i's message in round r_c, and by lines 25–28 of the updateSynchDetector() procedure, $Active^{r_c-1}_i[r'] \subseteq Active^{r_c}_j[r']$ and $Failed^{r_c-1}_i[r'] \subseteq Failed^{r_c}_j[r']$ for all $r' < r$.

Induction step: Let r_i be the first round $\geq r$ in which $p_i \in Active^{r_i}_j[r]$. If $r_i = r$, then p_j receives a message from p_i in round r and the claim is true.

If $r_i > r$, then p_j receives a message in round r_i from a process p_m such that $p_i \in Active^{r_i-1}_m[r]$. Since $p_i \in Active^{r_i-1}_m[r]$, we can apply the induction hypothesis with $r_c := r_i - 1$ and $r := r$ to obtain that $Active^r_i[r'] \subseteq Active^{r_i-1}_m[r']$ and $Failed^r_i[r'] \subseteq Failed^{r_i-1}_m[r']$, for all $r' < r$. Since p_j receives a message from p_m in round r_i, the claim follows by the same reasoning as in the base case above. □

The idea behind the third proposition is that if an estimate is held by some process at round r, then there exists at least one process which "carries" it in every previous round.

Proposition 3 (Carriers) *Let $r > 0$ and $p \in \Pi$. If p has estimate v at the end of round r, then for all rounds $0 \leq r' \leq r$, there exists a process $q^{r'} \in \mathit{Active}^r_p[r']$ such that $\mathit{est}^{r'-1}_{q^{r'}} = v$.*

Proof Assume that process p has estimate v at the end of round r, yet there exists a round $0 \leq r_0 \leq r$ such that no process with $\mathit{est}^{r_0-1} = v$ exists in $\mathit{Active}^r_p[r_0]$. First, notice that there has to exist a process s in $\mathit{Active}^r_p[r]$ such that $\mathit{est}^{r-1}_s = v$—this follows since a process may only adopt an estimate that has been proposed in the current round.

Let $r' < r$ be the minimum round such that there exists a process s in $\mathit{Active}^r_p[r' + 1]$ such that $\mathit{est}^{r'}_s = v$—the observation above ensures that such a round exists. As $s \in \mathit{Active}^r_p[r' + 1]$, Proposition 2 ensures that $\mathit{Active}^{r'}_s[r'] \subseteq \mathit{Active}^r_p[r']$. Since $\mathit{Active}^r_p[r']$ contains no processes with $\mathit{est}^{r'-1} = v$, it follows that $\mathit{Active}^{r'}_s[r']$ contains no processes with $\mathit{est}^{r'-1} = v$, and hence s has adopted estimate v at the end of r' without receiving any messages with estimate v, which contradicts the structure of the estimate-update mechanism. □

The next proposition proves that two processes with synchronous views see the same information, with a maximum delay of one round.

Proposition 4 (View Consistency) *Given processes p and q that see rounds $r_0 + 1, \ldots, r_0 + \ell + 1$ as synchronous, for all $r \in [r_0 + 1, r_0 + \ell]$, $\mathit{Active}^{r_0+\ell+1}_p[r + 1] \subseteq \mathit{Active}^{r_0+\ell+1}_q[r]$.*

Proof In order to simplify notation, we omit the superscript for the state variables that are seen from the end of $r_0 + \ell + 1$, e.g. $\mathit{Active}_p[r_0 + \ell] = \mathit{Active}^{r_0+\ell+1}_p[r_0 + \ell]$. We make the distinction when necessary.

Assume by contradiction that there exists a round $r \in [r_0 + 1, r_0 + \ell]$ and a process $s \in \mathit{Active}_p[r + 1] \setminus \mathit{Active}_q[r]$. Since $s \notin \mathit{Active}_q[r]$, no process x in $\mathit{Active}_q[r_0 + \ell + 1]$ can have $s \in \mathit{Active}^{r_0+\ell}_x[r]$. Therefore, $\forall \pi \in \mathit{Active}_q[r_0 + \ell + 1]$, $s \in \mathit{Failed}^{r_0+\ell}_\pi[r]$. However, since $|\mathit{Active}_p[r_0 + \ell + 1]| \geq n - t$, $|\mathit{Active}_q[r_0 + \ell + 1]| \geq n - t$ and $n - t > \frac{n}{2}$, p receives at least one message from a process in $\mathit{Active}_q[r_0 + \ell + 1]$ in round $r_0 + \ell + 1$. Since $\mathit{Failed}_p[r]$ is the union of all Failed sets p_i received, it follows that $s \in \mathit{Failed}_p[r]$. At the same time, $s \in \mathit{Active}_p[r + 1]$, and therefore p notices an asynchrony in round $r \in \{r_0 + 1, \ldots, r_0 + \ell\}$: contradiction. □

The next step is to show that in order for $k + 1$ estimates to be maintained in the system in a round, at least k failures have to occur in that round (these failures may be either process crashes, or messages not delivered in a timely manner because of asynchrony). More precisely, we identify one carrier that does not receive k messages which are received by at least one of the other carriers.

Proposition 5 *Let r be a round and $c_1, \ldots, c_{k+1}$ be processes such that $\mathit{est}^r_i < \mathit{est}^r_{i+1}, \forall i \in [1, k]$. Then there exists a process $c_\ell \in \{c_1, \ldots, c_{k+1}\}$ such that c_ℓ does not see k processes that were active in round r, i.e.*

$$\left| Failed^r_{c_\ell}[r] \cap \bigcup_{i \in [1,k+1]} Active^r_{c_i}[r] \right| \geq k.$$

Proof Processes $c_1, \dots, c_{k+1}$ are *carriers* for values $v_1, \dots, v_{k+1}$, respectively, at the end of round r. Proposition 3 ensures that there exist processes $q_1, \dots, q_{k+1}$ that are carriers for these values at the end of the previous round $r-1$ and $q_i \in Active_{c_i}[r]$, for all $i \in [1, k+1]$.[8] We prove that there exists an index j such that process c_j does not receive messages from any of the processes q_i with $i \neq j$.

Consider process c_{k+1}. Assuming that c_{k+1} receives a message from one of the processes q_i with $i \neq j$, it follows that c_{k+1} "sees" an estimate less than est_{k+1} in round r. In this case, the only possibility for c_{k+1} to stick to estimate est_{k+1} at the end of round r is for it to receive est_{k+1} in a message with $sFlag_i = \mathsf{true}$. Without loss of generality, assume that the message comes from process q_{k+1}. At this point, we turn our attention to process c_k. Again, there are two possibilities: process c_k adopted estimate est_k either because it received it in a message with $sFlag = \mathsf{true}$ (line 21), or adopts it in line 22, which means that it receives no messages with $sFlag = \mathsf{true}$, and no message with estimate $< est_k$. However, in the latter case we are done, since it means that c_k does not receive messages from any of the processes in $\{q_1, q_2, \dots, q_{k-1}, q_{k+1}\}$ which is enough to prove the claim.

Therefore, we still have to analyze the case when process c_k receives estimate v_k in a message with $sFlag = \mathsf{true}$. Again, without loss of generality, we assume that this message comes from process q_k. At this point, we are in the case where q_{k+1} has $sFlag = \mathsf{true}$ and q_k has $sFlag = \mathsf{true}$. Considering process c_{k-1}, we can apply the same rationale to obtain that process q_{k-1} necessarily has $sFlag = \mathsf{true}$. We proceed in this fashion to obtain that processes $q_{k-2}, \dots, q_1$ must have $sFlag = true$ as well. However, returning to process q_{k+1}, we obtain a contradiction, since if processes $q_1, q_2, \dots, q_k$ have $sFlag = \mathsf{true}$, process q_{k+1} cannot receive a message from any one of them (otherwise it will adopt an estimate $< est_{k+1}$ in line 21 of propose()).

Therefore, at least one process $c_j \in \{c_1, c_2, \dots, c_{k+1}\}$ has to fail all processes q_i with $i \neq j$ in round r. Since, by definition, $q_i \in Active_{c_i}[r]$, for all $i \in [1, k+1]$, this concludes the proof. □

At this point, we have gathered enough information to proceed with the final argument.

We show that the synchrony requirements on the views of $p_1, p_2, \dots, p_{k+1}$ imply that there exists a process q which they all perceive as active in round $r_0 + \lfloor t/k \rfloor + 2$. In fact, we show in Proposition 6 that q may be any process in P. This means that process q's message in round $r_0 + \lfloor t/k \rfloor + 2$ reaches all processes $p_1, p_2, \dots, p_{k+1}$, either directly or through a relay. This implies that the view of process q at the end of round $r_0 + \lfloor t/k \rfloor + 1$ has to be *consistent* with that of processes $p_1, p_2, \dots, p_{k+1}$, i.e. upon receiving q's message, no process in $\mathcal{P}$ notices an asynchrony in rounds $r_0 + 1, r_0 + 2, \dots, r_0 + \lfloor t/k \rfloor + 1$. Since processes $p_1, p_2, \dots, p_{k+1}$ all hold distinct estimates at the end of $r_0 + \lfloor t/k \rfloor + 3$, we show that q's view has to contain k new

[8] To simplify notation, we omit the superscript for the local variables, assuming that all such variables are seen from the end of round r.

failures in each round $r_0 + 1, \dots, r_0 + \lfloor t/k \rfloor$. This will imply that, in order to maintain a synchronous view, processes in $\mathcal{P}$ have to see $k \cdot \lfloor t/k \rfloor = t - (t \bmod k)$ failures in rounds $r_0 + 2, \dots, r_0 + \lfloor t/k \rfloor + 1$, which means that processes $p_1, \dots, p_{k+1}$ have at most $t \bmod k$ failures "left" at the end of round $r_0 + \lfloor t/k \rfloor + 1$. Finally, we apply Proposition 5 to obtain that at least one of the processes p_i has to see k new failures in round $r_0 + \lfloor t/k \rfloor + 3$, which leads to a contradiction, since the model ensures that all processes in $\mathcal{P}$ receive at least $n - t$ messages in every round.

We first show that there exists a process from which all processes p_i receive a message in round $r_0 + \lfloor t/k \rfloor + 2$, either directly or though a relay.

Proposition 6 (The common process) *Given processes* $p_1, p_2, \dots, p_{k+1}$ *as above, there exists a process* q *such that*

$$q \in \bigcap_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 2].$$

Proof Fix a process $q \in \{p_1, \dots, p_{k+1}\}$. For all processes $p_i \in \{p_1, \dots, p_{k+1}\}$, Proposition 4 implies that $Active_q[r_0 + \lfloor t/k \rfloor + 3] \subseteq Active_i[r_0 + \lfloor t/k \rfloor + 2]$. Since $q \in Active_q[r_0 + \lfloor t/k \rfloor + 3]$ (a process always receives messages from itself), we obtain that $q \in \bigcap_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 2]$. □

Note that in the following, we omit the subscript when denoting q's view and assume the $Active_q$ and $Failed_q$ sets are always seen from the end of round $r_0 + \lfloor t/k \rfloor + 1$. In other words, we are analyzing the view that process q broadcasts to processes in $\mathcal{P}$ at the beginning of round $r_0 + \lfloor t/k \rfloor + 2$.

The next proposition shows that process q defined above has to receive k *less* messages in each round $r_0 + 2, \dots, r_0 + \lfloor t/k \rfloor + 1$.

Proposition 7 *Given the process* q *as defined in Lemma 6, for all rounds* $r \in \{r_0 + 1, \dots, r_0 + \lfloor t/k \rfloor\}$,

$$|Active_q[r] \setminus Active_q[r+1]| \geq k.$$

Proof Proposition 3 ensures that at the beginning of round $r + 1$ there exist carriers $c_1, c_2, \dots, c_{k+1}$ for values $v_1, v_2, \dots, v_{k+1}$ respectively, where $est_i^r = v_i$ and $c_i \in Active_{p_i}^{r_m}[r + 1]$. (Recall that the round r_m has been defined as $r_0 + \lfloor t/k \rfloor + 3$.)

The key to proving the claim is to look at which of these carriers process q receives a message from in round $r + 1$. If process q receives no message from these carriers in $r + 1$, then we are done, since there are $k + 1$ carriers. Otherwise, we show that if process q receives a message from m such carriers (for $m \geq 1$), then it has to perceive at least $m - 1$ failures in round r just because it sees m distinct values propagated in the following round. Next, we show that q has to perceive a failure in round r for each of the other $k + 1 - m$ values whose carriers did not successfully send q a message in round $r + 1$. The final argument shows that the two sets of failures (the $m - 1$ corresponding to the "seen" carriers and the $k + 1 - m$ corresponding to the "unseen" ones) are necessarily distinct.

We start the formal argument by noting that processes $c_1, c_2, \ldots, c_{k+1}$ are necessarily in $Active_q[r]$: if $c_i \notin Active_q[r]$, since p_i receives q's message in $r_0 + \lfloor t/k \rfloor + 2$, process p_i perceives an asynchrony in round r. Next, denote by M the set of processes in the intersection $\{c_1, \ldots, c_{k+1}\} \cap Active_q[r+1]$. Let $m = |M|$, the cardinal of M, and let $\{s_1, \ldots, s_m\} = M$.

If $m = 0$, we have identified a set of at least $k+1$ processes $\{c_1, \ldots, c_{k+1}\}$ that are in $Active_q[r]$, but not in $Active_q[r+1]$ and we are done.

If $m > 0$, we can apply Proposition 5 to the processes (or the process) in M to obtain that there exists a process $s \in M$ such that $|Failed_s^r[r] \cap \bigcup_{j=1}^m Active_{s_j}^r[r]| \geq m-1$. Let F denote the set $Failed_s^r[r] \cap \bigcup_{j=1}^m Active_{s_j}^r[r]$, that is the set of processes that carrier s missed in round r. Since $s \in Active_q[r+1]$, by Proposition 2, $Failed_s^r[r] \subseteq Failed_q[r]$. Also, $F \cap Active_q[r+1] = \emptyset$ (otherwise, q notices an asynchrony in round r, which propagates to the processes p_i). On the other hand, since $s_j \in Active_q[r+1]$ for all $s_j \in M$, by applying Proposition 2 again we obtain that $F \subset Active_q[r]$. Therefore, the set F has at least $m-1$ processes and is in $Active_q[r] \setminus Active_q[r+1]$.

In order to find more processes in $Active_q[r] \setminus Active_q[r+1]$, we analyze the set of processes $G = \{c_1, c_2, \ldots, c_{k+1}\} \setminus Active_q[r+1]$, that is the set of carriers whose message process q did not receive. By similar considerations as above, these processes are elements of $Active_q[r] \setminus Active_q[r+1]$, and the cardinal of G is $k+1-m$. We show that $G \cap F = \emptyset$.

Assume for the sake of contradiction that there exists a process $c_i \in F \cap G$. Since $c_i \in F$, it follows that $c_i \in Failed_q[r]$. On the other hand, since $c_i \in G$, there exists a process $p_i \in \mathcal{P}$ such that $c \in Active_i^{r_m}[r+1]$, by the definition of c_i. However, since p_i receives a message from q in round $r_0 + \lfloor t/k \rfloor + 2 > r+1$, it follows by Proposition 2 that $c_i \in Active_i^{r_m}[r+1] \cap Failed_i^{r_m}[r]$, so p_i perceives an asynchrony in round $r \in \{r_0+1, \ldots, r_0 + \lfloor t/k \rfloor\}$, which contradicts the definition of process p_i.

Therefore $G \cap F = \emptyset$. The processes in $G \cup F$ have the property that they are in $Active_q[r]$, but not in $Active_q[r+1]$. The above claim ensures that $|G \cup F| \geq (k+1-m+m-1) = k$, therefore $|Active_q[r] \setminus Active_q[r+1]| \geq k$. □

The last result implies that $|Active_q[r_0 + \lfloor t/k \rfloor + 1]| \leq n - k \cdot \lfloor t/k \rfloor = n - t + t \bmod k$, therefore $|Failed_q[r_0 + \lfloor t/k \rfloor + 1]| \geq t - (t \bmod k)$. Note that since every process p_i receives a message from q in round $r_0 + \lfloor t/k \rfloor + 2$, no process p_i can receive a message in round $r_0 + \lfloor t/k \rfloor + 2$ from a process that q has failed in round $r_0 + \lfloor t/k \rfloor + 1$. More precisely, for all $i \in \{1, \ldots, k+1\}$, $Active_i^{r_0+\lfloor t/k \rfloor+3}[r_0 + \lfloor t/k \rfloor + 2] \subseteq Active_q^{r_0+\lfloor t/k \rfloor+1}[r_0 + \lfloor t/k \rfloor + 1]$.

This relation implies the following bound on the number of messages that processes in $\mathcal{P}$ may receive in round $r_0 + \lfloor t/k \rfloor + 2$:

$$\left| \bigcup_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 2] \right| \leq n - t + t \bmod k.$$

Next, we show that this number of active processes is not enough to maintain $k+1$ distinct values in the system in the remaining rounds $r_0 + \lfloor t/k \rfloor + 2$ and $r_0 + \lfloor t/k \rfloor + 3$. One way to see this is to first notice that the total number of messages received by processes in $\mathcal{P}$ may only decrease or remain the same, i.e. $\bigcup_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 2] \supseteq \bigcup_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 3]$ by Proposition 1. Then $|\bigcup_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 3]| \leq n - t + t \bmod k$. On the other hand, processes $p_1, p_2, \ldots, p_{k+1}$ have $k+1$ distinct estimates at the end of round $r_0 + \lfloor t/k \rfloor + 3$, therefore we can apply Proposition 5 to obtain that there exists a process $p_j \in \{p_1, \ldots, p_{k+1}\}$ such that $|Failed_j[r_0 + \lfloor t/k \rfloor + 3] \cap \bigcup_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 3]| \geq k$.

Then $|Active_j[r_0 + \lfloor t/k \rfloor + 3]| \leq |\bigcup_{i=1}^{k+1} Active_i[r_0 + \lfloor t/k \rfloor + 3] \setminus Failed_j[r_0 + \lfloor t/k \rfloor + 3]| \leq (n - t + t \bmod k) - k < n - t$, so process p_j receives less than $n - t$ messages in round $r_0 + \lfloor t/k \rfloor + 3$, a contradiction with the assumption that each process receives at least $n - t$ messages in every round.

The contradiction arises from the initial assumption that there exist $k+1$ processes with distinct estimates and synchronous views of rounds $r_0 + 1, \ldots, r_0 + \lfloor t/k \rfloor + 3$ at the end of round $r_0 + \lfloor t/k \rfloor + 3$. We conclude that the Elimination Lemma holds.

5.4 Improving the Algorithm

In fact, in some cases, processes can decide after seeing only $\lfloor t/k \rfloor + 3$ consecutive synchronous rounds: in brief, a process sets $sFlag = \mathsf{true}$ after seeing $\lfloor t/k \rfloor + 2$ synchronous rounds, and decides one round later under the same conditions as K4. In this case, however, the proof argument from the previous section works only if $\lfloor \frac{n-t+1}{k+1} \rfloor \geq 3k$, which translates approximately into $t \geq 3k^2$. In order to improve further, for example, to decide in $\lfloor t/k \rfloor + 2$ rounds, some new technique is needed. We believe that an approach similar to that of [2] in which estimates are sometimes *de-prioritized* can be used to obtain a matching algorithm.

6 Conclusion

We have presented a novel 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 by determining the cost of early decision in terms of worst-case round complexity. More generally, our simulation technique is applicable to any *decision task*, i.e. one in which a process can safely copy its decision from others. We believe that our technique can also be expressed in terms of the standard BG simulation [5]. In particular, instead of employing n simulators that agree through adopt-commit objects, we can use $k+1$ simulators that utilize BG-agreements to agree on the messages received in every round. Thus, one direction of future work is to extend our lower bound results to other families of tasks by encapsulating the Extended BG simulation [13]. Another direction is to fill the gap between the lower bound and the upper bound in eventually synchronous systems.

Proving distributed impossibility results and lower bounds often requires analysis of distributed executions, which has proven quite challenging (e.g., techniques involving algebraic topology). Moreover, there are a plethora of different models, multiplying the number of times each result needs to be re-proved. By contrast, distributed simulations offer the hope of deriving these results by direct reduction, thus basing the edifice of distributed computing on a few fundamental results. We believe that our results are one step toward developing just such a unified framework for distributed computation.

Acknowledgements We would like to thank Hagit Attiya, Keren Censor-Hillel, and the anonymous reviewers for their feedback on drafts of this paper.

Part of the work was performed as C. Travers was a Post-Doctoral Fellow at the Technion, Haifa, supported by the “Sam & Cecilia Neaman” Fellowship. Part of the work was performed as S. Gilbert was a Post-Doctoral Fellow at the Swiss Federal Institute of Technology, Lausanne, Switzerland.

References

1. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. J. ACM **40**(4), 873–890 (1993)
2. Alistarh, D., Gilbert, S., Guerraoui, R., Travers, C.: How to solve consensus in the smallest window of synchrony. In: Proceedings of the 22nd International Symposium on Distributed Computing (DISC), pp. 32–46 (2008)
3. Attiya, H., Rachman, O.: Atomic snapshots in $O(n\log n)$ operations. SIAM J. Comput. **27**(2), 319–340 (1998)
4. Biely, M., Schmid, U., Weiss, B.: Synchronous consensus under hybrid process and link failures. Theor. Comput. Sci. **412**(40), 5602–5630 (2011)
5. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 91–100 (1993)
6. Cerf, V., Icahn, R.: A protocol for packet network intercommunication. Comput. Commun. Rev. **35**(2), 71–82 (2005)
7. Charron-Bost, B., Schiper, A.: The heard-of model: computing in distributed systems with benign faults. Distrib. Comput. **22**(1), 49–71 (2009)
8. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
9. Chaudhuri, S., Herlihy, M., Lynch, N.A., Tuttle, M.R.: A tight lower bound for k-set agreement. J. ACM **47**(5), 912–943 (2000)
10. Dutta, P., Guerraoui, R.: The inherent price of indulgence. Distrib. Comput. **18**(1), 85–98 (2005)
11. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
12. Gafni, E.: Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract). In: Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 143–152 (1998)
13. Gafni, E.: The Extended BG simulation and the characterization of t-resiliency. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC) (2009)
14. Gafni, E., Guerraoui, R., Pochon, B.: The complexity of early deciding set agreement. From a static impossibility to an adaptive lower bound: the complexity of early deciding set agreement. SIAM J. Comput. **40**(1), 63–78 (2011)
15. Guerraoui, R., Herlihy, M., Pochon, B.: A topological treatment of early-deciding set-agreement. Theor. Comput. Sci. **410**(6–7), 570–580 (2009)
16. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
17. Keidar, I., Shraer, A.: Timeliness, failure-detectors, and consensus performance. In: Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 169–178 (2006)
18. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)

19. Raipin Parvédy, P., Raynal, M., Travers, C.: Strongly terminating early-stopping set agreement in synchronous systems with general omission failures. Theory Comput. Syst. **47**(1), 259–287 (2010)
20. Saks, M.E., Zaharoglou, F.: Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)
21. Widder, J., Schmid, U.: Booting clock synchronization in partially synchronous systems with hybrid process and link failures. Distrib. Comput. **20**(2), 115–140 (2007)
22. Yang, J., Neiger, G., Gafni, E.: Structured derivations of consensus algorithms for failure detectors. In: Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 297–308 (1998)