Distrib. Comput. (2005) 18(1): 85–98
DOI 10.1007/s00446-005-0124-9

SPECIAL ISSUE PODC

Partha Dutta · Rachid Guerraoui

The inherent price of indulgence

Received: October 2002 / Revised: July 2003 / Accepted: September 2004 / Published online: 12 April 2005

Abstract An indulgent algorithm is a distributed algorithm that tolerates asynchronous periods of the network when process crash detection is unreliable. This paper presents a tight bound on the time complexity of indulgent consensus algorithms.

We consider a round-based *eventually synchronous* model, and we show that any t-resilient consensus algorithm in this model, requires at least $t + 2$ rounds for a global decision even in runs that are *synchronous*. We contrast our lower bound with the well-known $t + 1$ round tight bound on consensus in the synchronous model. We then prove the bound to be tight by exhibiting a new t-resilient consensus algorithm in the eventually synchronous model that reaches a global decision at round $t + 2$ in every synchronous run. Our new algorithm is in this sense significantly faster than the most efficient indulgent algorithm we know of, which requires $2t + 2$ rounds in synchronous runs.

Our lower bound applies to round-based consensus algorithms with unreliable failure detectors such as $\diamond P$ and $\diamond S$, and our matching algorithm can be adapted to such failure detectors.

Keywords Fault tolerance · Distributed algorithms · Consensus time complexity

1 Introduction

1.1 Motivation

Indulgent algorithms are distributed algorithms that can tolerate asynchronous periods of the network when slow processes cannot be distinguished from those that have crashed [9]. These algorithms are indulgent towards their failure detector in the sense that, for an arbitrary period of time, failure detection mistakes are forgiven. This characteristic makes indulgent algorithms particularly attractive in systems with unpredictable processing and communication delays. We consider in this paper indulgent algorithms that deterministically solve the consensus problem [7, 12] in a message-passing distributed system with n processes: we denote by t the maximum number of processes that might fail and assume that processes can fail only by crashing.

It is well-known that indulgence entails a resilience price: [2] has shown that a majority of correct processes ($t < n/2$) is necessary for any consensus algorithm to tolerate unreliable failure detection,whereas non-indulgent algorithms can solve consensus even with a minority of correct processes. Does indulgence also entail a performance price? That is, does the unreliability of failure detection make indulgent consensus algorithms inherently less efficient than non-indulgent consensus algorithms?

We contribute in addressing this question by focusing on the performance of *synchronous* runs (i.e., runs in which failure detection is reliable) of consensus algorithms in an *eventually synchronous* model (an important class of algorithms that tolerate unreliable failure detection). We investigate whether synchronous runs of algorithms in eventually synchronous model are slower than runs of consensus algorithms specifically designed for a synchronous model. Besides scientific curiosity, investigating synchronous runs of indulgent consensus algorithms is interesting because, in many real systems, most runs are actually synchronous.

1.2 Model

We consider a crash-stop message-passing distributed system consisting of a set of $n \geq 3$ processes: $\Pi = \{p_1, p_2, \ldots, p_n\}$. A process executes the deterministic algorithm assigned to it until the algorithm terminates or the process (possibly) *crashes*. Processes do not recover from a crash. A *correct* process is a process that never crashes; all other processes are *faulty*. Every pair of processes can communicate through *send* and *receive* primitives, such that each message

This work is partially supported by the Swiss National Science Foundation (project number 510-207).

P. Dutta · R. Guerraoui (✉)
Distributed Programming Laboratory, EPFL, Lausanne, Switzerland
E-mail: {Partha.Dutta, Rachid.Guerraoui}@epfl.ch

is received at most once, no message is altered, and no message is received without having been sent.

We consider two round-based models: the well-known synchronous crash-stop model [13], which we denote by *SCS*, and an eventually synchronous model, denoted by *ES*. In both models, the computation proceeds in rounds with increasing round numbers starting from 1. If a process enters a round then it either completes the round or crashes. Each round consists of two phases: (1) in the send phase, the processes are supposed to send messages, timestamped with the current round number, to all other processes,[1] and (2) in the receive phase, the processes receive some messages and update their states accordingly. In the receive phase of any round k, if a process p_i does not receive the round k message from some other process p_j, then we say that p_i *suspects* p_j in round k. We say that a message m sent by a process p_i to p_j is *lost* if p_j never receives m in that run. We now describe the two models we consider.

1. In *SCS*, if a process p_i crashes in some round k, then any subset of the messages sent by p_i in that round may be lost, and the remaining messages sent by p_i are received in the same round. If p_i does not crash in round k, then every process which completes round k, receives the round k message from p_i.
2. In *ES*, the runs may be "asynchronous" for an arbitrary yet finite number of rounds but eventually become "synchronous". In *ES*, a message may be *delayed* for a finite number of rounds; i.e., received in a round higher than in which it was sent. More precisely, for every run in *ES*, the following properties hold:
   - (*t-resilience*) every process which completes any round k, receives round k messages from at least $n-t$ processes,
   - (*reliable channels*) messages sent from correct processes to correct processes are never lost but may be delayed for an arbitrary yet finite number of rounds, and
   - (*eventual synchrony*) there is an unknown but finite round number K such that, in every round $k \geq K$, (a) if a process p_i crashes in round k, then any subset of the messages sent by p_i in that round may be lost, and the remaining messages sent by p_i are received in the same round, and (b) if p_i does not crash in round k, then every process which completes round k, receives the round k message from p_i.

We say that a run in *ES* is *synchronous* if $K = 1$ in that run.[2] In a non-synchronous (or asynchronous) run, a process p_i may suspect another process p_j in some round k even if p_j has not crashed in round k. We call such suspicions *false suspicions*.

1.3 Time complexity of consensus

A consensus algorithm assists a set of processes to decide on a single value among the values proposed by the processes. We define consensus here using two primitives: propose($*$) and decide($*$). Each process is supposed to propose a value v from a known set V of values by invoking propose(v) and a process decides a value v by invoking decide(v). Consensus ensures the following properties: (1) (*validity*) if a process decides v then some process has proposed v, (2) (*uniform agreement*) no two processes decide differently,[3] and (3) (*termination*) every correct process eventually decides. Binary consensus is a variant of consensus in which $V = \{0, 1\}$.

We say that a run of a consensus algorithm in a round-based model (e.g., *SCS* or *ES*) achieves a *global decision* at round k if (1) all processes which ever decide in that run, decide at round k or at a lower round and (2) at least one process decides at round k. It is well-known that in *SCS* (1) every consensus algorithm has a run which requires $t+1$ rounds for a global decision (provided $t \leq n-2$) [13], and (2) the *FloodSet* algorithm of [13] solves consensus in *SCS* and achieves global decision at round $t+1$ in every run.

This paper studies the time-complexity of consensus algorithms in *ES*. From [7] we know that every consensus algorithm in *ES* has a run which takes an arbitrary number of rounds for every deciding process to decide (because a run in *ES* can remain "asynchronous" for an arbitrary number of rounds). Thus, we focus on synchronous runs of *ES*. As a measure of the time complexity of consensus algorithms in *ES*, we seek the round number k_{ES} such that: (1) every consensus algorithm in *ES* has a synchronous run which requires at least k_{ES} rounds for a global decision (i.e., every consensus algorithm in *ES* has a synchronous run in which some process decides at round k_{ES} or at a higher round), and (2) there is a consensus algorithm in *ES* which achieves a global decision at round k_{ES} or at a lower round in every synchronous run.

1.4 Contributions

This paper shows that $k_{ES} = t+2$. Roughly speaking, the price of indulgence is one round.

[1] For simplicity of presentation, we assume that processes are supposed to send messages to all other processes in every round. If such a message is not generated by the algorithm, the processes simply send dummy messages.

[2] A closer look at the synchronous runs of *ES* reveals that they provide slightly more guarantees than the runs in *SCS*: messages sent by a process p_i to correct processes, in the round in which p_i crashes, can only be delayed in synchronous runs of *ES*, whereas, such messages may be lost in a run of *SCS*. However, this only strengthens our result, as we are interested in the worst-case lower bound among synchronous runs.

[3] Although we prove our lower bound in the context of the uniform consensus problem, it immediately extends to the non-uniform version of the problem: [9] has shown that any indulgent algorithm that solves non-uniform consensus, also solves uniform consensus. Unless otherwise mentioned in this paper, consensus always refers to the uniform variant of the problem.

– First, we show that, for every consensus algorithm A in ES, among all synchronous runs of A, there is at least one run in which some process decides at round $t + 2$ or at a higher round, provided $0 < t < n/2$.[4] Our proof extends the technique of [1], used to prove the $t + 1$ round lower bound for consensus algorithms in SCS, to consensus algorithms in ES: indistinguishability of runs in our proof results from process crashes *as well as* from false suspicions.

  We also discuss how our lower bound can be extended to synchronous runs of an asynchronous round-based model enriched with unreliable failure detectors, such as $\diamond P$ and $\diamond S$. Furthermore, a variant of ES, which does not have the t-resilience property and in which all delayed messages are lost, is identical to the fail-stop basic round model of [6] (Sects. 3.1 and 3.2.1 of [6]), and trivial simplification of our lower bound proof in ES applies to that model of [6].
– Second, we show that our bound is tight by exhibiting a consensus algorithm in ES which achieves a global decision at round $t + 2$ in every synchronous run. It is a flooding algorithm that tries to detect false suspicions by exchanging the set of suspected processes and expedites decision whenever it detects the absence of false suspicions.

  We then explain how to modify our algorithm to rely on an asynchronous round-based model enriched with a $\diamond S$ failure detector. The resulting algorithm is significantly more efficient (in worst-case synchronous runs) than any other $\diamond S$-based consensus algorithm we know of. Our $\diamond S$-based algorithm achieves a global decision at round $t + 2$ in "synchronous runs". In contrast, the $\diamond S$-based consensus algorithm of [10], which used to be the most efficient in worst-case synchronous runs among the indulgent consensus algorithms we knew of, has a synchronous run which requires $2t + 2$ rounds for a global decision.

1.5 Roadmap

Section 2 presents our lower bound proof. Section 3 exhibits a consensus algorithm that achieves the bound. Section 4 relates ES with round-based asynchronous models enriched with unreliable failure detectors. Section 5 presents two extensions of our matching algorithm: a matching algorithm in a round-based asynchronous model enriched with failure detector $\diamond S$, and a simple optimization of our algorithm for failure-free synchronous runs. Finally, Sect. 6, discusses the case of fast early decision in synchronous runs, and fast eventual decision for runs which are synchronous after some round k.

[4] We exclude the following two cases in our result. (1) $t = 0$: in this case, processes can decide after exchanging proposal values in the very first round (say on the proposal value of p_1). (2) $t \geq n/2$: in this case, as we recalled earlier, there is no indulgent solution to consensus.

2 The lower bound

Proposition 1 *Let $0 < t < n/2$. Every consensus algorithm in ES, has a synchronous run in which some process decides at round $t + 2$ or at a higher round.*

Proof overview We assume by contradiction that there is a binary consensus algorithm A which globally decides at round $t+1$ in every synchronous run. The primary idea of the proof is that the indistinguishability of some synchronous runs from some non-synchronous runs, at the end of round $t + 1$ at some process, obstructs all synchronous runs from globally deciding in $t + 1$ rounds.

We use the traditional bivalency based technique of [7] to prove our lower bound result. To define the valency of a partial run (or a configuration) we consider only a subset of all runs that extend the partial run; namely, synchronous runs in which at most one failure occur in each round. First we consider only synchronous partial runs. We show that it is impossible to globally decide in one round from a bivalent partial run. Thus, to show a contradiction we need to construct a t-round bivalent partial run of A. Following [1], we start with a bivalent initial configuration and extend it to a $(t - 1)$-round bivalent partial run. It is easy to see that we cannot construct a bivalent t-round partial run by playing with only synchronous runs: this would contradict the $t + 1$ round tight bound on consensus in SCS. We need to introduce non-synchronous runs to maintain bivalency for an extra round.

We assume that all one round synchronous extensions of our $(t - 1)$-round bivalent partial run r_{t-1} are univalent. From this assumption we construct two synchronous extensions s^0 and s^1 of r_{t-1} whose decision values are 0 and 1, respectively. Informally speaking, to derive a contradiction we show how to construct two non-synchronous runs a^0 and a^1 such that (1) at the end of round $t+1$ some process p cannot distinguish s^0 from a^0, as well as s^1 from a^1 (and hence, p decides 0 in a^0 and 1 in a^1), and (2) the other processes can never distinguish a^0 from a^1. □

Proof Suppose by contradiction that there is a binary consensus algorithm A (possible proposal values are 0 and 1) in every synchronous run of which, any process which ever decides, decides at the end of round $t + 1$. We prove four lemmas (Lemma 2 to Lemma 5) on algorithm A. Lemma 5 contradicts Lemma 2.

Before stating and proving the lemmas we present some definitions and notations. We say that a run r of A is a *serial run* if r is a synchronous run and at most one process crashes in every round of r.[5] Clearly, as every serial run is a synchronous run, in every serial run of A, every process which ever decides, decides at the end of round $t + 1$.

An *l-round (serial) partial run* of A is a partial run of A which is identical to the first l rounds of some (serial)

[5] Note that, even in a synchronous run of ES, messages sent by a process p_i in the round in which p_i crashes, may be delayed for an arbitrary number of rounds.

run of A. Conversely, a run r of A is an *extension* of an l-round partial run r_l if the first l rounds of r is identical to r_l. Furthermore, if the extension r of r_l is a serial run, then we say r is a *serial extension* of r_l. A one-round (serial) extension of an l-round serial partial run r_l is an $(l+1)$-round (serial) partial run whose first l rounds are identical to r_l.

Note that the configuration of the system at the end of any partial run consists of the state of individual processes and the set of delayed messages in the communication channels.

We say that a k-round serial partial run r_k is 0-valent (1-valent) if the only decision value in all serial extensions of r_k is 0 (respectively, 1). A k-round serial partial run is univalent if it is either 0-valent or 1-valent; otherwise, it is bivalent. An initial configuration C_0 is 0-valent (1-valent) if the only possible decision value in all serial runs starting from C_0 is 0 (respectively, 1). An initial configuration is univalent if it is either 0-valent or 1-valent; otherwise, the initial configuration is bivalent.

Without loss of generality, we assume that in a round a process sends the same message to all processes: this message can be an array of messages where element j $(1 \leq j \leq n)$ of the array contains the original message intended for p_j. We denote the message sent by any process p_i at round k of run r by $m_r(i,k)$. $M_r(i,k)$ denotes the set of messages received by p_i at round k of run r. □

Lemma 2 *Every t-round serial partial run is univalent.*

Proof Suppose by contradiction that there is a t-round serial partial run r_t which is bivalent. Suppose that run r^0 is a serial extension of r_t such that no process crashes after round t. Without loss of generality, we assume that r^0 has decision value 0. Since run r^0 is serial, every process which ever decides in r^0, decides 0 at the end of round $t+1$. Furthermore, as r_t is bivalent, there is a serial run r^1 which has decision value 1: every process which ever decides in r^1, decides 1 at the end of round $t+1$. Notice that as both runsr^0 and r^1 are extensions of r_t, the processes cannot distinguish the runs at the beginning of round $t+1$, and therefore, the messages sent by any process at round $t+1$ are identical in both runs, i.e., $\forall p_l \in \Pi,\ m_{r^0}(l,t+1) = m_{r^1}(l,t+1)$.

Consider a process p_i which is correct in both runs r^0 and r^1($t < n/2$ implies that there is a process which is correct in both runs). $M_{r^0}(i,t+1)$ and $M_{r^1}(i,t+1)$ are the sets of messages received by p_i at round $t+1$ of r^0 and r^1, respectively. Since p_i is correct, p_i must decide (at round $t+1$ of serial runs r^0 and r^1). To decide at round $t+1$, p_i must be able to distinguish r^0 from r^1 at round $t+1$, which implies that $M_{r^0}(i,t+1) \neq M_{r^1}(i,t+1)$. As no process crashes at round $t+1$ of r^0, we have $M_{r^1}(i,t+1) \subset M_{r^0}(i,t+1)$.

Now we construct a one-round asynchronous extension of r_t, say $a^{0,1}$, as follows. Round $t+1$ of $a^{0,1}$ is identical to round $t+1$ of r^0, except that p_i makes some false suspicions and p_i receives $M_{r^1}(i,t+1)$ instead of $M_{r^0}(i,t+1)$ (which is possible because $M_{r^1}(i,t+1) \subset M_{r^0}(i,t+1)$), i.e., p_i is the only process which can distinguish the first $t+1$ rounds of r^0 from the partial run $a^{0,1}$. Process p_i cannot distinguish the partial run $a^{0,1}$ from the first $t+1$ rounds of r^1, and hence, p_i decides 1 at the end of $a^{0,1}$. Consider a process p_j which is correct in r^0 and distinct from p_i. The assumption that $0 < t < n/2$ implies that $t+2 \leq n$, i.e., there are two correct processes in any run. Clearly, p_j cannot distinguish the first $t+1$ rounds of r^0 from $a^{0,1}$. Thus, p_j decides 0 in $a^{0,1}$. Consider a run r whose first $t+1$ rounds are identical to $a^{0,1}$. Clearly, r violates consensus agreement: a contradiction. □

Lemma 3 *There is an initial configuration which is bivalent.*

Proof Suppose by contradiction that every initial configuration is univalent. Consider the initial configurations C_0 and C_n in which all processes propose 0 and 1, respectively. From consensus validity, it follows that C_0 is 0-valent and C_n is 1-valent. Define C_i (for every i such that $0 < i < n$) as the initial configuration in which every process p_j such that $j \leq i$ proposes 1 and all other processes propose 0. Consider a serial run r_{C_i} starting from C_i $(0 \leq i < n)$ in which process p_{i+1} crashes initially and other processes decide $d \in \{0,1\}$ at round $t+1$. Notice that, even if the initial configuration in r_{C_i} is changed to C_{i+1}, the decision value remains d (because p_{i+1} crashes before sending any messages in r_{C_i}). Thus, if $C_i (0 \leq i < n)$ is d-valent then C_{i+1} is also d-valent.

Using the above result and a simple induction we can show that, if C_0 is 0-valent, then so is C_n: a contradiction. □

Lemma 4 *(From [1]) There is a $(t-1)$-round serial partial run which is bivalent.*

Proof The proof is by induction on round number k $(0 \leq k < t-1)$.
Base Step. From Lemma 3 it follows that there is a 0-round serial partial run which is bivalent.
Induction Hypothesis. There is a k-round serial partial run r_k which is bivalent $(0 \leq k < t-1)$.
Induction Step. By contradiction, we assume that every one-round serial extension of r_k is univalent.

Suppose that every one-round serial extension of r_k is univalent. Let r^0_{k+1} be a $(k+1)$-round serial partial run which is an extension of r_k such that no process crashes at round $k+1$. Without loss of generality, we can assume that r^0_{k+1} is 0-valent. Since r_k is bivalent, there is a $(k+1)$-round serial partial run r^*_{k+1} which is an extension of r_k and which is 1-valent. There must be exactly one process p'_1 which crashes in round $k+1$ of r^*_{k+1} and there is a (possibly empty) set of processes $\{p'_2, \ldots, p'_m\}$ that can distinguish r^0_{k+1} from $r^*_{k+1} (0 \leq m-1 < n)$: i.e., the processes which received a message from p'_1 at round $k+1$ of r^0_{k+1} and did not receive a message from p'_1 at round $k+1$ of r^*_{k+1}.

Consider the following $(k+1)$-round serial partial runs $r^1_{k+1}, \ldots, r^m_{k+1}$ such that: (1) r^1_{k+1} is identical to r^0_{k+1}, except that in r^1_{k+1}, p'_1 crashes at round $k+1$, though the round $k+1$ message sent from p'_1 to other processes are received

at round $k+1$. (2) $r^j_{k+1}(2 \leq j \leq m)$ is identical to r^0_{k+1} except that, in r^j_{k+1}, p'_1 crashes at round $k+1$ such that the $(k+1)$-round messages sent by p'_1 to $\{p'_2, \ldots, p'_j\}$ are lost (although non-crashed processes in $\Pi \backslash \{p'_2, \ldots, p'_j\}$ receive the $(k+1)$-round message sent by p'_1 in the same round). Now consider the following two claims which contradict the fact that r^*_{k+1} is 1-valent.

4.1. If $r^i_{k+1}(0 \leq i < m)$ is 0-valent then so is r^{i+1}_{k+1}: partial runs r^i_{k+1} and r^{i+1}_{k+1} differ only in the state of process p'_{i+1} at the end of round $k+1$. Consider a one-round serial extension r_{k+2} of r^i_{k+1} in which p'_{i+1} crashes at the beginning of round $k+2$ (before sending any message in round $k+2$) and no other process crashes in round $k+2$. Also, consider a one-round serial extension r'_{k+2} of r^{i+1}_{k+1} in which p'_{i+1} crashes at the beginning of round $k+2$ (if $p'_{i+1} = p'_1$ then it has already crashed in round $k+1$) and no other process crashes in round $k+2$.[6] Obviously, at the end of round $k+2$ no non-crashed process can distinguish r_{k+2} from r'_{k+2}. Consider serial extensions of r_{k+2} and r'_{k+2} in which no process crashes after round $k+2$. Since, $k+2 < t+1$, at the end of round $t+1$, the two runs are identical at all non-crashed processes. Thus the decision values are the same in both extensions. So, if r^i_{k+1} $(0 \leq i < m)$ is 0-valent, then r^{i+1}_{k+1} is also 0-valent. It follows that r^m_{k+1} is 0-valent.

4.2. r^*_{k+1} is 0-valent: serial partial runs r^*_{k+1} and r^m_{k+1} are identical. Therefore, r^*_{k+1} is 0-valent: a contradiction. □

Lemma 5 *There is a t-round serial partial run which is bivalent.*

Proof Suppose by contradiction that every t-round serial partial run is univalent. From Lemma 4 we know that there is a bivalent $(t-1)$-round serial partial run, which we denote by r_{t-1}. Let r^0_t be a one-round serial extension of r_{t-1} such that no process crashes at round t. Without loss of generality, we can assume that r^0_t is 0-valent. Since r_{t-1} is bivalent, there is a one-round serial extension r^*_t of r_{t-1} which is 1-valent. There must be exactly one process p'_1 which crashes in round t of r^*_t and there is a (possibly empty) set of processes $\{p'_2, \ldots, p'_m\}$ that can distinguish r^0_t from r^*_t $(0 \leq m-1 < n)$: i.e., the processes which received a message from p'_1 at round t of r^0_t and did not receive a message from p'_1 at round t of r^*_t.

Consider the following t-round serial partial runs $r^1_t, \ldots, r^m_t$ such that: (1) r^1_t is identical to r^0_t, except that in r^1_t, p'_1 crashes at round t, though the round t message sent from p'_1 to other processes are received at round t. (2) $r^j_t(2 \leq j \leq m)$ is identical to r^0_t, except that in r^j_t, p'_1 crashes at round t such that the t-round messages sent by p'_1 to $\{p'_2, \ldots, p'_j\}$ are lost (although non-crashed processes in $\Pi \backslash \{p'_2, \ldots, p'_j\}$ receive the t-round message sent by p'_1 in the same round). Now consider the following two claims which contradict the fact that r^*_t is 1-valent.

5.1. If $r^i_t(0 \leq i < m)$ is 0-valent then so is r^{i+1}_t: the proof is given in the following subsection. The claim implies that r^m_t is 0-valent.

5.2. r^*_t is 0-valent: partial runs r^m_t and r^*_t are identical. Therefore r^*_t is 0-valent: a contradiction. □

Proof of Claim 5.1

The proof of Claim 4.1 does not work for the present case. To see why, notice that in Claim 4.1, $k+1$ processes may have crashed in serial partial run r^{i+1}_{k+1}. Since $k+1 < t$ (in Lemma 4), we can crash one more process in any extension of r^{i+1}_{k+1}, which is necessary to show that r^i_{k+1} and r^{i+1}_{k+1} have the same valency. However, in the present case, t processes may have already crashed in r^{i+1}_t.

Proof Suppose by contradiction that r^i_t is 0-valent and r^{i+1}_t is 1-valent. Serial partial runs r^i_t and r^{i+1}_t differ only in the state of p'_{i+1} at the end of round t. There are two cases: (1) $p'_{i+1} = p'_1$, or (2) $p'_{i+1} \neq p'_1$.

If $p'_{i+1} = p'_1$ (i.e., p'_{i+1} is up at the end of $r^i_t = r^0_t$ but crashes in $r^{i+1}_t = r^1_t$, and in both partial runs, no message is lost in round t), then we reach a contradiction as follows. From the definition of a serial run, we know that at most t processes can crash in r^{i+1}_t. Since r^i_t and r^{i+1}_t are identical except for state of p'_{i+1}(p'_{i+1} crashes in r^{i+1}_t but not in r^i_t), at most $t-1$ processes could have crashed in r^i_t. So, we can construct a serial run r' by extending r^i_t in which p'_{i+1} crashes at the beginning of round $t+1$ (before sending any messages in that round). From round $t+1$ onwards, no process can ever learn whether r' is a serial extension of r^i_t or a serial extension of r^{i+1}_t. Consequently, if r^i_t is 0-valent then so is r^{i+1}_t: a contradiction.

Consider the case when $p'_{i+1} \neq p'_1$. Process p'_{i+1} is the only process which can distinguish r^i_t from r^{i+1}_t at the end of round t: p'_{i+1} receives a t-round message from p'_1 in r^i_t and does not receive a t-round message from p'_1 in r^{i+1}_t.

In the following we construct two synchronous and three asynchronous runs. Figure 1 depicts round t and $t+1$ of the five runs. For clarity of presentation, we only indicate the messages which assist in distinguishing the constructed runs from each other.

First we construct two synchronous runs s^1 and s^0 in which p'_{i+1} decides different values.

s^1: This serial run is an extension of r^{i+1}_t in which no process crashes after round t. Since the partial run r^{i+1}_t is 1-valent and s^1 is a serial run, p'_{i+1} decides 1 at the end of round $t+1$.

[6] Note that p'_{i+1} may crash at the beginning of round $k+2$ in r'_{k+2} because, by the definition of serial runs, at most $k+1 < t$ processes can crash in the first $k+1$ rounds. $k+1 < t$ because the induction is done over $0 \leq k < t-1$.

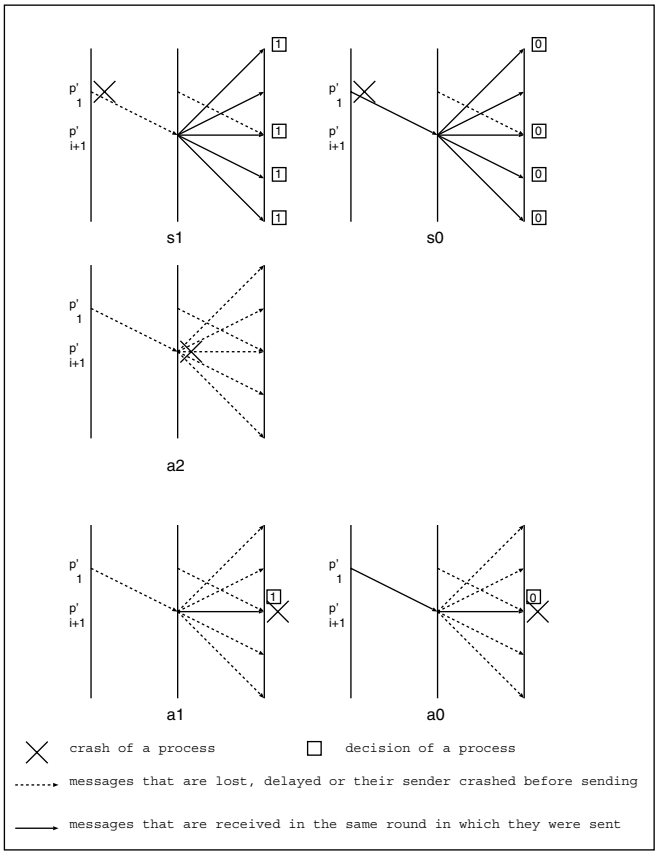

Fig. 1 Rounds t and $t+1$ of the runs in the proof of Claim 5.1

s^0: This serial run is an extension of r_t^i in which no process crashes after round t. Since the partial run r_t^i is 0-valent and s^0 is a serial run, p'_{i+1} decides 0 at the end of round $t+1$.

We now construct three $(t+1)$-round asynchronous runs a^2, a^1, and a^0. In the construction of these runs we maintain the property of *ES* that in each round of each run, every non-crashed process receives at least $n-t$ messages sent in that round.

a^2: This asynchronous run is constructed as follows for each round k:
- $k \leq t-1$: The first $t-1$ rounds of a^2 are identical to those of s^1.
- $k = t$: No process crashes in this round. Unlike s^1, p'_1 does not crash in round t of this run. Every process distinct from p'_1 receives the same set of messages as in round t of s^1. (In other words, p'_1 is falsely suspected by $\{p'_2, \ldots, p'_{i+1}\}$ and the corresponding messages are delayed.) Process p'_1 receives messages from all non-crashed processes at round t.
- $k \geq t+1$: Process p'_{i+1} crashes before sending any messages in round $t+1$. There are no other crashes or false suspicions at round $t+1$ or at a higher round. All the delayed messages of round t are received at round $t+2$. (Notice that no new delayed message is generated after round t.) From the termination property of consensus, there is a round $k' \geq t+1$ such that a^2 reaches a global decision at k'.

Observations: (1) At the end of round t of a^2, only p'_1 can distinguish the first t rounds of a^1 from the first t rounds of s^1. (2) At most $t-1$ processes have crashed in the first t rounds of a^2. To see why, notice that the first $t-1$ rounds of a^2 is identical to the first $t-1$ rounds of s^1. As s^1 is a serial run, at most $t-1$ processes can crash in the first $t-1$ rounds of s^1. As no process crashes in round t of a^2, by the end of round t in a^2, at most $t-1$ processes have crashed.

a^1: This asynchronous run is constructed as follows for each round k:
- $k \leq t$: The first t rounds a^1 are identical to those of of a^2.
- $k \geq t+1$: Unlike in a^2, p'_{i+1} does not crash in round $t+1$. However, the processes that are distinct from p'_{i+1} falsely suspect p'_{i+1} in round $t+1$ and p'_{i+1} falsely suspects p'_1 in round $t+1$. Process p'_{i+1} crashes before sending any message in round $t+2$. There are no other crashes or false suspicions in round $t+2$ or at a higher round. The delayed messages of round t are received at round $t+2$, and the delayed messages generated in round $t+1$ (i.e., the messages sent by p'_{i+1}) are received in round $k'+1$. (Round k' is defined in the description of run a^2.)

We claim that p'_{i+1} cannot distinguish a^1 from s^1 at the end of round $t+1$. Notice that the first $t-1$ rounds of s^1, a^2, and a^1 are identical. At the end of round t, a^2 and a^1 are identical, and only p'_1 can distinguish a^2 from s^1. Thus, at the end of round t, only p'_1 can distinguish a^1 from s^1. Now consider the messages received by p'_{i+1} in round $t+1$ of a^1 and s^1. Process p'_{i+1} does not receive any message from p'_1 (due to the crash of p'_1 in s^1, and due to the false suspicion of p'_1 by p'_{i+1} in a^1). Since, no process distinct from p'_1 can distinguish a^1 from s^1 at the end of round t, p'_{i+1} receives identical sets of messages in round $t+1$ of both a^1 and s^1. Thus p'_{i+1} cannot distinguish a^1 from s^1 at the end of round $t+1$, and hence, decides 1 at the end of round $t+1$.

a^0: This asynchronous run is constructed as follows for each round k:
- $k \leq t-1$: The first $t-1$ rounds of a^0 are identical to those of s^0.
- $k = t$: No process crashes in this round. Unlike in s^0, p'_1 does not crash in round t of a^0. But, every process distinct from p'_1 receives the same set of messages as in round t of s^0. (In other words, p'_1 is falsely suspected by $\{p'_2, \ldots, p'_i\}$ and the corresponding messages are delayed.) Process p'_1 receives messages from all non-crashed processes at round t.
- $k \geq t+1$: Processes distinct from p'_{i+1} falsely suspect p'_{i+1} in round $t+1$ and p'_{i+1} falsely suspects

```
at process p_i
 1: procedure propose(v_i)
 2:   est_i ← v_i; k_i ← 1; msgSet_i ← ∅; Halt_i ← ∅; nE_i ← v_i; vc_i ← v_i; decided_i ← false
 3:   Phase 1
 4:   while k_i ≤ t + 1 do                                   {rounds 1, ..., t + 1}
 5:     send(ESTIMATE, k_i, est_i, Halt_i) to all
 6:     wait until received messages of round k_i
 7:     compute()
 8:     k_i ← k_i + 1
 9:   Phase 2
10:   if |Halt_i| > t then                                   {round t + 2}
11:     nE_i ← ⊥
12:   else
13:     nE_i ← est_i
14:   send(NEWESTIMATE, t + 2, nE_i) to all
15:   wait until received messages of round t + 2
16:   if every received(NEWESTIMATE, t + 2, nE) has nE ≠ ⊥ then
17:     vc_i ← any one of the nE values received
18:     decide(vc_i); decided_i ← true                      {Decision}
19:   else if received any (NEWESTIMATE, t + 2, nE') message s.t. nE' ≠ ⊥ then
20:     vc_i ← nE'
21:   k_i ← k_i + 1                                         {round t + 3}
22:   if decided_i then
23:     send (DECIDE, t + 3, vc_i) to Π\p_i
24:   else
25:     propose_C(vc_i)
26:   return                                                {return from propose(*)}

27: upon receiving (DECIDE, k', x) from p_j do
28:   wait until k_i ≥ k'
29:   if ¬decided_i then
30:     stop propose_C(); decide(x); send (DECIDE, k', x) to Π\p_i   {Decision}
31:   return                                                {return from propose(*)}

32: procedure compute()
33:   Halt_i ← Halt_i ∪ {p_j | (p_i received(ESTIMATE, k_i, *, Halt_j) from p_j s.t. p_i ∈ Halt_j) or
      (p_i did not receive round k_i message from p_j)}
34:   msgSet_i ← {(ESTIMATE, k_i, *, Halt_j) | p_i received(ESTIMATE, k_i, *, Halt_j) from p_j ∉
      Halt_i}
35:   est_i ← Min{est | (ESTIMATE, k_i, est, *) ∈ msgSet_i}
```

Fig. 2 The consensus algorithm A_{t+2}

p'_1 in round $t+1$. Process p'_{i+1} crashes before sending any message in round $t+2$. There are no other crashes or false suspicions in round $t+2$ or at a higher round. The delayed messages of round t are received in round $t+2$, and the delayed messages generated in round $t+1$ (i.e., the messages sent by p'_{i+1}) are received in round $k'+1$.

We make the following two claims.

- Process p'_{i+1} cannot distinguish a^0 from s^0 at the end of round $t+1$. Notice that only process p'_1 can distinguish a^0 from s^0 at the end of round t, and p'_{i+1} falsely suspects p'_1 in round $t+1$. Thus, at round $t+1$, p'_{i+1} receives identical sets of messages in a^0 and s^0. Thus p'_{i+1} cannot distinguish a^0 from s^0 at the end of round $t+1$, and hence decides 0 at the end of round $t+1$.
- At the end of round k', processes distinct from p'_{i+1} cannot distinguish a^2, a^1, and a^0. To see why observe that the first $t-1$ rounds of the three runs are identical. At the end of round t, the runs a^2, a^1 and a^0 (1) differ at process p'_{i+1}: p'_{i+1} falsely suspects p'_1 in round t of a^2 and a^1, but receives the round t message from p'_1 in a^0, and consequently (2) also differ in one delayed message in the communication channels: there is a round t delayed message from p'_1 to p'_{i+1} in a^2 and a^1, which is not delayed in a^0. The message sent by p'_{i+1} to other processes at round $t+1$ are received in round $k'+1$, in a^1 and a^0, and p'_{i+1} crashes before sending any messages at round $t+1$ in a^2. Thus, the processes that are distinct from p'_{i+1} cannot distinguish the three runs before round $k'+1$. However, every process which decides in a^2, decides by round k' in a^2. Thus, every process distinct from p'_{i+1}, which decides in any of the three runs, does so by round k' and decide the same value in all the three runs (because it cannot distinguish between the runs before round $k'+1$). Clearly, either a^1 or a^0 violates uniform agreement because p'_{i+1} decides 1 in a^1 and 0 in a^0; a contradiction. □

3 A matching consensus algorithm

Figure 2 presents a consensus algorithm A_{t+2} in ES when $0 < t < n/2$, which achieves the lower bound of Proposition 1. Namely, besides solving consensus, A_{t+2} satisfies the following property:

Fast decision In every synchronous run of A_{t+2}, any process which ever decides, decides by round $t+2$.

The algorithm assumes an underlying independent consensus module C,[7] accessed through procedure propose$_C(*)$, and which decides through procedure decide(∗). The fast decision property is achieved by A_{t+2} regardless of the time complexity of C. More precisely, our algorithm assumes:

1. the *ES* model with $0 < t < n/2$;
2. that no process ever suspects itself;
3. an independent consensus algorithm C in *ES* when $0 < t < n/2$;
4. that the set of proposal values in a run is a totally ordered set; e.g., each process p_i can tag its proposal value with its index i and then the values can be ordered based on this tag.

3.1 Basic idea

Our algorithm A_{t+2} is a variant of the *FloodSetWS*[8] algorithm of [3], modified for exchanging and tracking false suspicions. Algorithm A_{t+2} has two phases: Phase 1 lasts the first $t+1$ rounds and Phase 2 involves round $t+2$ and the underlying consensus algorithm C. In Phase 1, the processes exchange their estimates of the decision (initialized to the proposal values) and every process updates its estimate to the minimum of all estimates seen in the round. The primary objective of repeating this exchange for $t+1$ rounds is to converge towards the same estimate at all processes. However, this may be hindered by false suspicions, i.e., processes may have different estimates at the end of Phase 1. Therefore, the algorithm tries to detect the false suspicions to ensure the following *elimination property*: given any two processes which complete Phase 1, either both processes have the same estimate value or at least one of them detects a false suspicion.

At the beginning of Phase 2, the processes compute their new estimate as follows: if a process detects a false suspicion, then its new estimate is set to $\perp$; otherwise, the new estimate is the estimate value at the end of Phase 1. The processes exchange their new estimate values in round $t+2$. Due to the elimination property of Phase 1, in every run, the number of distinct new estimate values different from $\perp$ is at most one. If a process receives only non-$\perp$ new estimate values in round $t+2$, then it decides on any non-$\perp$ value received. Otherwise, achieving a decision is delegated to algorithm C: due to the consensus termination property of C, at every correct process, procedure propose$_C(*)$ eventually invokes decide(∗).

3.2 Description

Each process is supposed to invoke procedure propose(∗) with its proposal value as a parameter, and the procedure progresses in rounds of message exchanges. After receiving messages in any round k (in Phase 1), the processes invoke procedure compute() to update their local states. The algorithm tries to achieve consensus in the first $t+2$ rounds. However, if a process does not decide at round $t+2$, it invokes the underlying consensus algorithm C.

Every process p_i maintains the following variables:

k_i: the current round number,
est_i: the estimate of p_i which is set to the minimum value seen by p_i until round k_i, initialized to the proposal value v_i,
$Halt_i$: the set of processes p_j such that, in the present round or in a lower round, p_i suspected p_j or p_j suspected p_i,
nE_i: the new estimate of p_i computed at the beginning of round $t+2$,
vc_i: the proposal value for the underlying consensus algorithm C, initialized to the proposal value v_i, and
$decided_i$: a boolean indicating whether process p_i has decided.

Phase 1 In this phase, which consists of the first $t+1$ rounds, the processes exchange ESTIMATE messages containing est and $Halt$. On receiving the messages at round k, p_i updates its variables (by invoking the procedure compute()) as follows:

1. $Halt_i$ is the set of processes such that for every $p_j \in Halt_i$, p_i suspected p_j (i.e., p_i did not receive a message from p_j) in round k or in a lower round, or p_j suspected p_i (i.e., $p_i \in Halt_j$) in round $k-1$ or in a lower round.
2. $msgSet_i$ is the set of messages received by p_i at round k such that the senders of the messages are not in $Halt_i$.
3. est_i is updated to the minimum est value in $msgSet_i$.

Phase 2 We say that p_i detects a false suspicion in the first $t+1$ rounds if $|Halt_i| > t$ at the beginning of round $t+2$. In fact, if $|Halt_i| > t$ at the beginning of round $t+2$ then either p_i falsely suspected some other process or some other process falsely suspected p_i. To see why, suppose that $|Halt_i| > t$. If there is a process $p_j \in Halt_i$ such that $p_i \in Halt_j$ then, obviously p_j falsely suspected p_i. If no such p_j is in $Halt_i$, then p_i suspected more than t processes, and hence, at least one of the suspicions is a false suspicion. In Lemma 13, we give a detailed proof of the claim that $|Halt_i| > t$ at the beginning of round $t+2$ implies that there is a false suspicion in the run.

In Phase 2, the processes first compute nE as follows. If p_i does not detect any false suspicion in the first $t+1$ rounds, then p_i sets nE_i to the minimum est value p_i has seen (i.e., the latest est_i value). Otherwise, nE_i is set to $\perp$. The processes exchange nE in round $t+2$. If p_i receives only non-$\perp$ nE values, then p_i decides on any of the nE values received, and in round $t+3$, p_i sends a DECIDE message with the decision value to other processes and returns from the invocation. Otherwise, either p_i receives some $nE' \neq \perp$

[7] The algorithm C can be any round-based $\diamond P$ or $\diamond S$ consensus algorithm (e.g., the one based on $\diamond S$ in [2]) transposed to the *ES* model.

[8] (Algorithm FloodSetWS assumes *perfect* failure detection (P) and achieves global decision at round $t+1$ in every run. It is itself inspired by the *FloodSet* consensus algorithm of [13] in *SCS*.)

and sets vc_i to nE', or every nE value received by p_i is $\perp$ and vc_i retains its initial value, v_i. Subsequently, in round $t+3$, p_i invokes propose$_C(vc_i)$ which eventually decides.

If p_i receives a DECIDE message from round $t+3$, then p_i waits until it reaches round $t+3$, and (if p_i has not decided yet then) p_i decides on the decision value received.

3.3 The elimination property of A_{t+2}

For presentation simplicity, we introduce the following notation. Given any variable var_i at process p_i, we denote by $var_i[k_i]$ the value of var_i immediately after the completion of procedure compute() at round k_i, i.e., at the end of line 7 ($1 \leq k_i \leq t+1$). We assume that there exists a symbol *undefined* which is distinct from any possible value of the variables in the algorithm A_{t+2}. If p_i crashes before completing the procedure compute() of round k_i, then for every variable var_i at p_i, we define $var_i[k_i] = undefined$. Similarly, if p_i completes compute() at round k_i, we define $senderMS_i[k_i]$ as the set of processes which have sent the messages which are in $msgSet_i[k_i]$, and $senderMS_i[k_i]$ is *undefined* if p_i crashes before completing round k_i. If p_i sends any NEWESTIMATE message (line 14), we denote the value of nE_i sent in that message by $nE_i[t+2]$. If p_i crashes before sending any NEWESTIMATE message, then $nE_i[t+2] = undefined$.

Lemma 6 (Elimination) *If there are two distinct processes p_x and p_y such that* (1) *p_x and p_y sent* NEWESTIMATE *messages,* (2) $nE_x[t+2] \neq \perp$, *and* (3) $nE_y[t+2] \neq \perp$, *then* $nE_x[t+2] = nE_y[t+2]$.

Proof Suppose by contradiction that there is a run r_{diff} of A_{t+2} and two distinct processes p_x and p_y such that in run r_{diff}: (1) $nE_x[t+2] = c \neq \perp$, (2) $nE_y[t+2] = d \neq \perp$, and (3) $c \neq d$. We prove five lemmas (Lemma 7 to Lemma 11) on run r_{diff}. Lemma 11 contradicts Lemma 8.

Without loss of generality we can assume that $c < d$. For run r_{diff} of A_{t+2} we define the set C_k as follows:

- C_0 is the set of processes whose proposal values are less than or equal to c.
- For every k such that $1 \leq k \leq t+1$, $C_k = \{p_j | est_j[k] \leq c \text{ or } est_j[k] = undefined\}$. In other words, C_k consists of the processes which either crash before completing compute() in round k or complete compute() in round k with $est \leq c$. □

From the definition of C_k, we can immediately make the following two observations about r_{diff}.

Observation O1 $|C_0| \geq 1$. Since $nE_x[t+2] = c$, some process must have proposed c.

Observation O2 For any k such that $0 \leq k \leq t$, $C_k \subseteq C_{k+1}$. Suppose by contradiction that $p_i \in C_k$ and $p_i \notin C_{k+1}$. Since $p_i \notin C_{k+1}$, p_i completes compute() in round $k+1$ with $est > c$, and therefore, completes compute() in round k if $k \geq 1$ (or completes line 2 if $k = 0$). Since $p_i \in C_k$, p_i completes round k with $est \leq c$ if $k \geq 1$ (or completes line 2 with $est \leq c$ if $k = 0$). Thus, p_i sends round $k+1$ message with $est \leq c$. Recall that we assume that a process never suspects itself. Therefore, $p_i \notin Halt_i[k+1]$ and p_i receives round $k+1$ message from itself. Consequently, while updating est_i in compute() of round $k+1$, p_i updates est_i to a value less than or equal to c; a contradiction.

Lemma 7 *For any k such that $1 \leq k \leq t+1$, and for any process p_l, if $senderMS_l[k] \neq undefined$ then sender $MS_l[k] = \Pi - Halt_l[k]$.*

Proof Suppose $senderMS_l[k] \neq undefined$. There are the following two cases to consider for every process $p_m \in \Pi$:

- $p_m \in Halt_l[k]$: from line 34, the message from p_m to p_l is not in $msgSet_l[k]$. Hence, $p_l \notin senderMS_l[k]$.
- $p_m \notin Halt_l[k]$: from line 33, p_l received the round k message from p_m. Thus, the message from p_m to p_l is in $msgSet_l[k]$ and $p_l \in senderMS_l[k]$. □

Lemma 8 $|C_t| \leq t$.

Proof Suppose by contradiction that $|C_t| > t$. Consider any process $p_m \in C_t$. Either p_m has completed compute() of round t with $est \leq c$ or p_m has crashed before completing compute() of round t. If p_m sends a message msg in round $t+1$, then $est \leq c$ in msg. Now consider the messages received by process p_y in round $t+1$. (Recall that $nE_y[t+2] = d > c$.) If $msgSet_y[t+1]$ contains a message from p_m, then from line 35 and line 13, $nE_y[t+2] \leq c$, a contradiction. Therefore, $msgSet_y[t+1]$ does not contain any message from p_m (i.e., $p_m \notin senderMS_y[t+1]$), and therefore, from Lemma 7, $p_m \in Halt_y[t+1]$. In other words, $\forall p_m \in C_t$, $p_m \in Halt_y[t+1]$, i.e., $C_t \subseteq Halt_y[t+1]$. Therefore, $|Halt_y[t+1]| \geq |C_t| > t$. It follows from line 10 that $nE_y[t+2]$ is $\perp$; a contradiction. □

Lemma 9 *$p_x \in C_{t+1}$ and $p_x \notin C_{t-1}$.*

Proof Notice that $nE_x[t+2] = c \neq \perp$ implies that $est_x[t+1] = c$ (line 13). Thus, from the definition of C_{t+1} it follows that $p_x \in C_{t+1}$.

For the next part of the proof, suppose by contradiction that $p_x \in C_{t-1}$. Since p_x completes compute() in round $t+1$ (because $nE_x[t+2] = c$), p_x also completes compute() in round $t-1$. Therefore, $est_x[t-1] \leq c$, and if p_x sends a message msg in round t, then $est \leq c$ in msg. Consider any process $p_m \in \Pi \backslash C_t$. From the definition of C_t, we know that $est_m[t] > c$. Therefore, $msgSet_m[t]$ does not contain any estimate message from p_x. (Otherwise, on receiving $est \leq c$ from p_x, p_m sets est_m to a values less than or equal to c.) Therefore, from Lemma 7 it follows that $p_x \in Halt_m[t]$. Consequently, if p_m sends a message msg in round $t+1$ then, $p_x \in Halt$ in msg.

From the algorithm it follows that, in round $t+1$, if p_x receives a msg from p_m then p_x includes p_m in $Halt_x$ (because $p_x \in Halt$ in msg sent by p_m), and if p_x does not

receive any message from p_m, then from line 33, we know that p_x includes p_m in $Halt_x$. It follows that, in either case, $\forall p_m \in \Pi \backslash C_t$, $p_m \in Halt_x[t+1]$, i.e., $\Pi \backslash C_t \subseteq Halt_x[t+1]$.

From Lemma 8 it follows that $|\Pi \backslash C_t| \geq n - t > t$ (recall that $t < n/2$). So, $|Halt_x[t+1]| > t$. From line 10 it follows that, if $|Halt_x[t+1]| > t$ then $nE_x[t+2] = \perp$; a contradiction. □

Lemma 10 *For all k such that* $0 \leq k \leq t-1$*:* $C_k \subset C_{k+1}$ *(i.e.,* $C_k \subseteq C_{k+1}$ *and* $C_k \neq C_{k+1}$*).*

Proof Consider any k such that $0 \leq k \leq t-1$. Recall from Observation O2 that $C_k \subseteq C_{k+1}$. Thus, either $C_k \subset C_{k+1}$ or $C_k = C_{k+1}$. Suppose by contradiction that there is a round k' such that $0 \leq k' \leq t-1$ and $C_{k'} = C_{k'+1}$. We make the following observation:

Observation O3 $\forall p_j \notin C_{k'+1}, C_{k'+1} \subseteq Halt_j[k'+1]$. Consider any process $p_j \notin C_{k'+1}$. From the definition of $C_{k'+1}$, $est_j[k'+1] > c$. Therefore, $msgSet_j[k'+1]$ does not contain an (ESTIMATE, $k'+1, *, *$) message from any process in $C_{k'}$: otherwise, $est_j[k'+1]$ must be less than or equal to c and $p_j \in C_{k'+1}$. Therefore, from Lemma 7, $C_{k'} \subseteq Halt_j[k'+1]$. Since $C_{k'} = C_{k'+1}$, $C_{k'+1} \subseteq Halt_j[k'+1]$.

For every $1 \leq k \leq t+1$, we define $aliveC_k$ as the set of processes which complete compute() in round k with $est \leq c$. We also define, $aliveC_0 = C_0$. From the definitions, $aliveC_k \subseteq C_k$.

Claim 10.1 $\text{aliveC}_{t+1} \subseteq C_{k'+1}$.

Proof We prove the claim by induction on the round number $s (k'+1 \leq s \leq t)$.

Base case $(s = k'+1)$ From the definition of $aliveC$, it immediately follows that $aliveC_{k'+1} \subseteq C_{k'+1}$.

Induction hypothesis $(k'+1 \leq s \leq t)$ $aliveC_s \subseteq C_{k'+1}$.

Induction Step. We need to show that $aliveC_{s+1} \subseteq C_{k'+1}$. Suppose by contradiction that there is a process $p_j \in aliveC_{s+1}$ such that $p_j \notin C_{k'+1}$. Thus, p_j completes compute() in round $s+1$ with $est \leq c$. Since $p_j \notin C_{k'+1}$, and from the induction hypothesis, $aliveC_s \subseteq C_{k'+1}$, it follows that $p_j \notin aliveC_s$. Thus, p_j completes compute() in round s with $est > c$. Since p_j completes compute() in round $s+1$ with $est \leq c$, there is a message with $est \leq c$ in $msgSet_j[s+1]$ from some process p_m. Obviously, p_m completes round s with $est \leq c$, and hence, $p_m \in aliveC_s$. From the induction hypothesis, it follows that $p_m \in C_{k'+1}$.

Since $p_j \notin C_{k'+1}$, from Observation 3, it follows that $C_{k'+1} \subseteq Halt_j[k'+1] \subseteq Halt_j[s+1]$. Thus, $p_m \in C_{k'+1} \subseteq Halt_j[s+1]$. However, a message from p_m is in $msgSet_j[s+1]$; a contradiction with Lemma 7. □

Proof of Lemma 10 continued. Now we show that, for $0 \leq k' \leq t-1$, $C_{k'+1} \subseteq C_{t-1}$ (Observation O4). If $k' \leq t-2$, then it follows from Observation O2 that $C_{k'+1} \subseteq C_{t-1}$ (because $k'+1 \leq t-1$). If $k' = t-1$, then from the definition of k', $C_{t-1} = C_t$, and thus, $C_{k'+1} = C_t \subseteq C_{t-1}$.

Since $aliveC_{t+1} \subseteq C_{k'+1}$ (from Claim 10.1), $C_{k'+1} \subseteq C_{t-1}$ (from Observation O4), and $p_x \notin C_{t-1}$ (from Lemma 9), it follows that $p_x \notin aliveC_{t+1}$. However, as $nE_x[t+2] = c$, p_x completed round $t+1$ with $est \leq c$, i.e., $p_x \in aliveC_{t+1}$, a contradiction. □

Lemma 11 $|C_t| \geq t+1$.

Proof From Lemma 10 we have for every k such that $0 \leq k \leq t-1$, $|C_{k+1}| - |C_k| \geq 1$. We know from Observation O1 that $|C_0| \geq 1$. Therefore, $|C_t| \geq t+1$. □

3.4 Correctness of A_{t+2}

The proof of the validity and the termination properties of the algorithm are straightforward. We now prove the agreement and the fast decision properties of A_{t+2}.

Lemma 12 (Agreement) *No two processes decide differently.*

Proof If no process ever decides, then agreement trivially holds. Consider the lowest round k in which some process decides. Say p_i decides v in round k. If $k > t+2$ then no process decides in the first $t+2$ rounds and hence no process sends a DECIDE message in line 23. Therefore, the agreement property follows from the agreement property of C. So, we consider the case when $k = t+2$. (From the algorithm, $t+2$ is the lowest round in which a process can decide.)

Since p_i decides v at round $t+2$, every message received by p_i in round $t+2$ has $nE \neq \perp$ and p_i received at least one message with $nE = v$. From the property of the *ES* model, there are at least $n - t > n/2$ messages received by p_i in round $t+2$. Therefore, at least a majority of the processes send NEWESTIMATE messages with $nE \neq \perp$. Furthermore, from Lemma 6 (elimination property), for every NEWESTIMATE message, $nE \in \{v, \perp\}$. Any process which completes round $t+2$ receives at least a majority of NEWESTIMATE messages (because $n - t > n/2$), and therefore, receives at least one message with $nE = v$. Consequently, if a process decides at round $t+2$ then this process decides v and sends a DECISION message with decision value v in round $t+3$, and if a process invokes $\text{propose}_C(*)$ then the invocation value is v. By the validity property of algorithm C, no process can decide on a value different from v. □

Lemma 13 (Fast Decision) *In every synchronous run of* A_{t+2}*, any process which ever decides, decides by round* $t+2$.

Proof Suppose by contradiction that there is a synchronous run in which some process completes round $t+2$ without deciding. Then it follows from line 16 that some process p_m has sent a NEWESTIMATE message with $nE = \perp$. Therefore in that synchronous run, $|Halt_m[t+1]| > t$ (line 10).

For every k such that $1 \leq k \leq t+1$, let us define $H[k]$ as $\{p_i | \exists p_j, p_i \in Halt_j[k]\}$. We define $H[0] = \emptyset$. First we show the following claim for any synchronous run. □

Claim 13.1 In any synchronous run, if any process p_i is in $H[t+1]$ then p_i has crashed before completing round $t+1$.

Proof We prove the claim by induction on the round number k $(0 \leq k \leq t)$.

Base Case ($k = 0$). Obvious, because $H[0] = \emptyset$.

Induction Hypothesis ($0 \leq k \leq t$) If a process p_i is in $H[k]$ then p_i crashes before completing round k.

Induction Step We need to show that if a process p_i is in $H[k+1]$ then p_i crashes before completing round $k+1$. Suppose by contradiction that there is a process p_i which completes round $k+1$ and $p_i \in H[k+1]$. Thus, there is a process p_j such that $p_i \in Halt_j[k+1]$. Notice that, since both p_i and p_j complete round k, it follows from the induction hypothesis that $p_i, p_j \notin H[k]$. Thus, $p_j \notin Halt_i[k]$, and hence, the message sent by p_i in round $k+1$ has $Halt_i$ such that $p_j \notin Halt_i$. Since $p_i \notin H[k]$, it follows that $p_i \notin Halt_j[k]$. Furthermore, as the run is synchronous and p_i does not crash in round $k+1$, p_j receives round $k+1$ message from p_i. Thus, from line 33, $p_i \notin Halt_j[k+1]$, a contradiction. □

Proof of Lemma 13 continued Since, $Halt_m[t+1] \subseteq H[t+1]$, $|H[t+1]| > t$. From Claim 13.1 it follows that more than t processes have crashed before completing round $t+1$, a contradiction. □

4 Extending the scope of the lower bound

In this section we show that our lower bound can be extended to asynchronous round-based models enriched with unreliable failure detectors. First, we define such models, and then discuss how to simulate such a model from *ES*.

An asynchronous round-based model where t processes may crash can be defined as follows. In each round, every process is supposed to send messages to all other processes, wait for $n-t$ messages sent in that round, and updates its state according to messages received [8]. The model may have reliable channels: messages sent from a correct process to a correct process is eventually received. In each round, a process may wait for $n-t$ messages without blocking because at least $n-t$ processes are correct in each run.

Informally speaking, a failure detector [2] is a distributed oracle which provides some information to each process about the crash of other processes. Each process has a local failure detector module. A failure detector is unreliable if it may provide any possible output (in its range) for an arbitrary yet finite period of time [9]. An example is eventually perfect failure detector $\diamond P$ [2] which outputs a set of suspected processes at each process such that (1) (*strong completeness*) eventually every process that crashes is permanently suspected by every correct process, and (2) (*eventual strong accuracy*) there is a time after which correct processes are not suspected by any correct process. The eventually strong failure detector $\diamond S$ differs from $\diamond P$ in its accuracy property: (*eventual weak accuracy*) there is a time after which some correct process is never suspected by any correct process.

In a round of an asynchronous round-based model enriched with a failure detector with the eventual strong completeness property, a process may also wait for messages from all processes not suspected by the local failure detector module. In this round-based model, we say that a run is synchronous if the messages received in each round satisfies the guarantees of synchronous runs in *ES*.

To simulate a round-based model enriched with $\diamond P$ or $\diamond S$ from *ES*, we give a possible output of the failure detector for every run in *ES*. This is done as follows: (1) at the beginning of the run, the simulated output is set to $\emptyset$, (2) on receiving messages of round k in *ES*, the simulated failure detector output is changed to the set of processes from which no message was received in round k of *ES*. To see why the properties of $\diamond P$ for instance hold, consider the round k such that (1) no message is delayed in round k or in a higher round, and (2) every faulty process crashes before starting round k. (Such a round exists from the property of *ES* and the from definition of faulty processes.) After round k, correct processes (1) never receive current round messages from faulty processes (since all faulty processes crash before starting round k), and (2) receive messages from all correct processes in every round (since no message is delayed after round k). Consider a time t such that all correct processes start round k before time t. It is easy to see that in our simulated failure detector output, after time t, every correct process permanently suspects every faulty process and does not suspect any correct process.

5 Extending the algorithm

5.1 A $\diamond S$-based algorithm

Our algorithm A_{t+2} can be easily transformed to a consensus algorithm that uses a $\diamond S$ failure detector [2, 10]. We denote the new algorithm by $A_{\diamond S}$ which is obtained by the following modifications to A_{t+2}: (1) substitute the underlying consensus algorithm C by any $\diamond S$-based consensus algorithm C' (e.g., that of [2]), and (2) modify line 6 and line 15 of Fig. 2 as shown in Fig. 3.

The correctness of $A_{\diamond S}$ is easy to verify, since consensus termination is ensured by the presence of at least

6: **wait until** ($\forall p_j \in \Pi$, received(ESTIMATE, k_i, $*$, $*$) from p_j or $p_j \in \diamond S_{p_i}$) **and** (received(ESTIMATE, k_i, $*$, $*$) from at least $n - t$ processes)
15: **wait until** ($\forall p_j \in \Pi$, received(NEWESTIMATE, $t + 2$, $*$) from p_j or $p_j \in \diamond S_{p_i}$) **and** (received(NEWESTIMATE, $t + 2$, $*$) from at least $n - t$ processes)

Fig. 3 Modifications for using $\diamond S$

$n - t$ correct processes, and the termination property of C'. More interestingly, $A_{\diamond S}$ retains the fast decision property of A_{t+2} because this property is relevant only in synchronous runs where the synchrony guarantees are much stronger than those of either *ES* or $\diamond S$-based asynchronous rounds.

5.2 Optimizing the failure-free case

In practice, failure-free runs are most prevalent among synchronous runs. There are many indulgent consensus algorithms in the literature which are optimized for failure-free runs in which the system is "well-behaved" [11]. These algorithms decide in two rounds if there are no failures and the underlying failure detector does not make any false suspicions. It has been shown in [11] that two rounds is a lower bound for global decision in such well-behaved runs.

Our algorithms A_{t+2} and $A_{\diamond S}$ can be easily improved to achieve a global decision at round 2 in every failure-free synchronous run. After receiving messages in round 2, if any process p_i is certain that there were no suspicions in round 1 (i.e., p_i receives round 2 messages from each of the n processes with $Halt = \emptyset$) then p_i decides immediately on any est value received, and sends a DECIDE message with the decision value to other processes in round 3. Otherwise, if p_i does not detect any suspicion at round 1 (i.e., p_i does not receive round 2 messages from all n processes, however, every round 2 message received by p_i has $Halt = \emptyset$) then p_i sets the proposal variable vc_i for the underlying consensus algorithm C to any estimate value received. Figure 4 describes the modification more precisely: the six lines in Fig. 4 are inserted between line 6 and line 7 of Fig. 2.

It is straightforward to see that Fig. 4 performs the required optimization without violating any of the consensus properties or the fast decision property. Suppose for instance that some process p_i decides d at round 2. To see why consensus agreement is not violated, notice that p_i decides in line 6.5 only if there has been a complete exchange of estimate messages at round 1 (i.e., no process suspected any process). As the proposal values form a totally ordered set, every ESTIMATE message at round 2 has the same est value d (where d is precisely the minimum of all proposed values), and therefore, every message sent at round 2 is (ESTIMATE, 2, d, $\emptyset$). Thus, the only possible decision value at round 2 is d, and every process p_j which completes round 2 without deciding, sets both vc_j and est_j to d. Therefore, any process which decides at round $t + 2$, decides d and any process which invokes propose$_C(*)$, does so with value d. Agreement is obvious.

6 Concluding remarks

Our lower bound result immediately implies a lower bound on early decision in synchronous runs of *ES*; i.e., for every f such that $1 \leq f \leq t$, every consensus algorithm in *ES*, with $0 < t < n/2$, has a synchronous run with at most f crashes, in which some process decides at round $f + 2$ or at a higher round. To see why, suppose by contradiction that there is an $f \leq t$ and a consensus algorithm A in *ES* (where at most t processes may crash) which globally decides by round $f + 1$ in every synchronous run with at most f crashes. Since $f \leq t$, A is also a consensus algorithm in *ES* where at most f processes may crash. Thus, in *ES* where at most f processes may crash, every synchronous run of A globally decides by round $f + 1$, a contradiction with Proposition 1. For $f \leq t - 2$, this lower bound also immediately follows from the $f+2$ round lower bound on consensus in *SCS* [4, 11]. However, whether this early decision bound in synchronous runs is tight was an open question (except for the failure-free case where it is known to be tight [11]). In a recent paper [5], we have shown that the $f + 2$ bound is indeed tight for all $f \leq t$.

Our $t + 2$ round bound for synchronous runs also raises the question of eventual fast decision. Recall that, in *ES*, for each run r in *ES*, there is a round K such that, in every round $k \geq K$, (a) if a process p_i crashes in round k, then any subset of the messages sent by p_i in that round may be lost, and the remaining messages sent by p_i are received in the same round, and (b) if p_i does not crash in round k, then every process which completes round k, receives the round k message from p_i. We say that such a run r *is synchronous after round* $K - 1$. Thus, a synchronous run in *ES* is synchronous after round 0. A simple modification of the proof of Proposition 1 implies that,[9] for every f such that $0 \leq f \leq t$, every consensus algorithm in *ES*, with $0 < t < n/2$, has a run: (1) that is synchronous after round k, (2) that has at most f crashes after round k, and (3) where some process decides at round $k + f + 2$ or at a higher round.

Whether the above bound is tight is an open question. In the following, we partially answer the question by describing a consensus algorithm that matches the bound for $0 \leq t < n/3$. Closing the gap for $n/3 \leq t < n/2$ is an open problem.

An efficient consensus algorithm when $t < n/3$ In Fig. 5, we present a new consensus algorithm A_{f+2} in *ES* for $t < n/3$. A_{f+2} is an optimized version of the second leader-based algorithms of [14], which we denote by A_{MR}. Algorithm A_{f+2} has the following fast eventual decision property: for every f such that $0 \leq f \leq t$, if a run becomes synchronous after round k, and there are f crashes after round k, then the run globally decides by round

[9] Informally speaking, the modification is done as follows: (1) redefine bivalency by considering only runs that are synchronous after round k, (2) construct a k-round asynchronous partial run which is bivalent, and (3) repeat the original proof after round k.

```
6.1: if k = 2 then
6.2     if every received (ESTIMATE, 2, est, Halt) message has Halt = ∅ then
6.3:        vc_i ← any one of the est values received
6.4:        if received(ESTIMATE, 2, est, ∅) message from all processes in Π then
6.5:           decide(vc_i) decided_i ← true                          {Decision}
6.6:           send (DECIDE, 3, vc_i) to Π\p_i; return                {return from propose(*)}
```

Fig. 4 Optimizing A_{t+2} for failure-free synchronous runs

```
at process p_i
 1: procedure propose(v_i)
 2:    est_i ← v_i; k_i ← 0; msgSet_i ← ∅
 3:    while(true) do
 4:       k_i ← k_i + 1
 5:       send(ESTIMATE, k_i, est_i) to all
 6:       wait until received messages of round k_i
 7:       if received any (DECIDE, k', est') such that k' ≤ k_i then
 8:          est_i ← est'; decide(est_i); break                        {decision}
 9:       msgSet_i ← set of messages of round k_i with the n − t lowest sender ids
10:       if every message in msgSet has the same est (say est') then
11:          est_i ← est'; decide(est_i); break                        {decision}
12:       if there are at least n − 2t messages in msgSet_i with the same est (say est') then
13:          est_i ← est'
14:       else
15:          est_i ← Min{est | (ESTIMATE, k_i, est) ∈ msgSet_i}
16:    send(DECIDE, k_i + 1, est_i) to Π\p_i; return                  {return from propose(*)}
```

Fig. 5 The consensus algorithm A_{f+2}

$k + f + 2$.[10] A_{f+2} is based on the following observation made in A_{MR}: when $t < n/3$, in a collection S of n values (where values may be repeated), if some value v appears $n - t$ times, then in *every* collection consisting of at least $n - t$ values from S, v appears at least $n - 2t$ times, and all other values appear less than $n - 2t$ times.

We now give a brief description of A_{f+2}. Consider any process p_i. Upon proposing some value v, p_i adopts v as the estimate est of the decision value. In each round, p_i exchanges its est, until p_i crashes or decides. A process sends two types of messages: an ESTIMATE message that contains the most recent est of the process, and a DECIDE message that contains the decision value of the process. Upon receiving the messages of round k, p_i first checks whether it has received any DECIDE message from round k or from a lower round, and if so, decides on the decision value received. Otherwise, among the messages received in round k, p_i selects $n - t$ messages with the lowest $n - t$ sender ids. (In other words, p_i arranges the messages in the ascending order of their respective senders' process ids, and then selects the first $n - t$ messages.) The set of these $n - t$ messages is denoted by $msgSet_i$ at p_i. If every message in $msgSet_i$ contains the same est (say est') then p_i decides est'. Otherwise, if some est value appears at least $n - 2t$ times in $msgSet_i$ then p_i adopts that value as est_i. If no est value appears at least $n - 2t$ times, then p_i adopts the minimum est value in $msgSet_i$ as est_i. Upon deciding, a process sends its decision value to other processes in the next round.

Correctness of A_{f+2} The validity property of the algorithm is rather straightforward. Thus we focus on the agreement, termination, and the fast eventual decision property.

Lemma 14 (Agreement) *No two processes decide differently.*

Proof Consider the lowest round k at which some process decides. A process can decide either in line 8 or in line 11. If a process decides at line 8 of round k then some process has sent a DECIDE message at round k or at a lower round, and hence, some process has decided at round $k - 1$ or at a lower round; a contradiction with the definition of k. Thus every process which decides at round k, decides in line 11.

Let p_i be a process which decides some value v in line 11 of round k. From the algorithm it follows that p_i received at least $n-t$ messages in round k with $est = v$. Since $t < n/3$, in round k, every process which completes round k, receives at least $n - 2t$ messages with $est = v$, and the number of messages received with $est \neq v$ is less than $n - 2t$. Thus every process which completes round k, either decides v or adopts v as est. A simple induction shows that processes can not have an estimate value (and hence a decision value) distinct from v in a higher round. □

Lemma 15 (Fast eventual decision) *Consider a run of algorithm A_{f+2} which is synchronous after round k. If there*

[10] We would like to point out that such a run of A_{MR} would require $k + 2f + 2$ rounds to globally decide. To translate the leader-based algorithm A_{MR} to *ES*, we simply need to implement an *eventual leader* primitive in *ES*, which can be done as follows: (1) every process p_i sends messages to all processes in every round, (2) p_i initially sets its variable *leader* to p_1, and (3) on receiving messages of a round k in *ES*, p_i sets its variable *leader* to the process with the minimum process id, among the senders of messages received by p_i in round k.

are f crashes ($0 \leq f \leq t$) after rounds k, then the run globally decides by round $k + f + 2$.

Proof Suppose by contradiction that there is a run of the algorithm that is synchronous after round k, f crashes occur after round k, and some process p_j completes round $k+f+2$ without deciding. There are the following two cases.

- Some process decides before round $k + f + 2$. Let $k' \leq k + f + 1$ be the lowest round in which some process decides. Let some process p_i decides v in round k'. As in Lemma 14, we can show that p_i receives at least $n - t$ messages with $est = v$, and every process which completes round k', either does so with $est = v$ or decides v. Thus, every process which sends a message in round $k' + 1$ either sends an ESTIMATE message with $est = v$ or sends a decision message with decision value v. Consequently, every process which completes round $k' + 1$ either receives a decision message or receives the same est value in every received message. In either case, the process decides in round $k' + 1 \leq k + f + 2$, a contradiction.
- No process decides before round $k + f + 2$. Since the algorithm is synchronous after round k and there are f crashes after round k, there is at least one round k', among the $f + 1$ rounds from $k + 1$ to $k + f + 1$, in which no crash occurs, and hence, every process which completes round k' receives the same set of messages. Thus the $msgSet$ at every process which completes round k' is the same (say $msgSet'$), and from the algorithm, $msgSet'$ contains exactly $n - t$ messages. If $msgSet'$ contains no two distinct estimate values, then all processes which complete round k', decide at the end of the round. Else, if some value v occurs at least $n - 2t$ times in $msgSet'$ then no other distinct value can occur $n - 2t$ times in $msgSet'$, and thus every process which completes round k', adopts v as est. If no value appears at least $n - 2t$ times in $msgSet'$, then every process which completes round k, adopts the minimum value in $msgSet'$ as its est. Thus, either there is a global decision in round k', or every process which completes round k', completes the round with the same est. In the second case, every process which completes round $k' + 1$, receives the same est value in every received message, and hence, decides. Thus every process which decides, decides by round $k' + 1 \leq k + f + 2$. □

Lemma 16 (Termination) *Every correct process eventually decides.*

Proof Consider any run r of A_{f+2}. We know that for every run in ES, there is round $k \geq 0$, such that, r is synchronous after round k. Since at most t processes may fail in a run, from Lemma 15 it follows that r globally decides by round $k + t + 2$. □

Acknowledgements We thank Petr Kouznetsov and Bastian Pochon for their helpful comments on earlier drafts of the paper. We are also grateful to the reviewers who helped us improve the presentation of the paper.

References

1. Aguilera, M.K., Toueg, S.: A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. Inform. Proces. Lett. **71**(3–4), 155–158 (1999)
2. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
3. Charron-Bost, B., Guerraoui, R., Schiper, A.: Synchronous system and perfect failure detector: solvability and efficiency issues. In: Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN), pp. 523–532 New York (2000)
4. Charron-Bost, B., Schiper, A.: Uniform consensus harder than consensus. DSC Technical Report 2000-28, Department of Communication Systems, Swiss Federal Institute of Technology, Lausanne (2000)
5. Dutta, P., Guerraoui, R., Pochon, B.: Tight bounds on early local decisions in uniform consensus. In: Proceedings of the 17th International Conference on Distributed Computing (DISC), Sorrento, Italy (2003)
6. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
7. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
8. Gafni, E.: Round-by-round fault detectors: Unifying synchrony and asynchrony. In: Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC-17), pp. 143–152 Puerto Vallarta, Mexico (1998)
9. Guerraoui, R.: Indulgent algorithms. In: Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC-19), pp. 289–298 Portland, OR (2000)
10. Hurfin, M., Raynal, M.: A simple and fast asynchronous consensus protocol based on a weak failure detector. Distribut. Comput. **12**(4), 209–223 (1999)
11. Keidar, I., Rajsbaum, S.: A simple proof of the uniform consensus synchronous lower bound. Inform. Process. Lett. **85**(1), 47–52 (2003); A preliminary version appeared in SIGACT News **32**(2), 45–63 (2001)
12. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Languages Syst. **4**(3), 382–401 (1982)
13. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
14. Mostefaoui, A., Raynal, M.: Leader-based consensus. Parallel Proce. Lett. **11**(1), 95–107 (2001)