Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Algorithms 51 (2004) 15–37

Journal of Algorithms

www.elsevier.com/locate/jalgor

Uniform consensus is harder than consensus

Bernadette Charron-Bost [a,*] and André Schiper [b]

[a] *Laboratoire d'Informatique LIX, École Polytechnique, 91128 Palaiseau cedex, France*
[b] *Faculté Informatique et Communications, École Polytechnique Fédérale, CH-1015 Lausanne, Switzerland*

Received 15 June 2001

Abstract

We compare the consensus and uniform consensus problems in synchronous systems. In contrast to consensus, uniform consensus is not solvable with byzantine failures. This still holds for the omission failure model if a majority of processes may be faulty. For the crash failure model, both consensus and uniform consensus are solvable, no matter how many processes are faulty. In this failure model, we examine the number of rounds required to reach a decision in the consensus and uniform consensus algorithms. We show that if uniform agreement is required, one additional round is needed to decide, and so uniform consensus is also harder than consensus for crash failures. This is based on a new lower bound result for the synchronous model that we state for the uniform consensus problem. Finally, an algorithm is presented that achieves this lower bound.
© 2003 Elsevier Inc. All rights reserved.

Keywords: Distributed algorithm; Synchronous model; Failures; Consensus; Uniform consensus; Time complexity; Early deciding algorithms

1. Introduction

The problem of reaching agreement in a distributed system in the presence of failures is a fundamental problem of both practical and theoretical importance. One version of this problem, called *consensus*, considers that each process starts with an initial value drawn from some domain V and all non faulty processes must decide on the same value. Moreover, if the initial values are the same, say v, then the only possible decision value for a non faulty process is v. Processors in the system are liable to fail by halting prematurely

[*] Corresponding author.
E-mail address: charron@lix.polytechnique.fr (B. Charron-Bost).

0196-6774/$ – see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jalgor.2003.11.001

(*crash failures*), by omitting to send or receive messages when they should (*omission failures*), or by exhibiting arbitrary behaviors (*byzantine failures*).

For many applications, the agreement condition of consensus, namely "no two non faulty processes decide differently", is inadequate as it does not restrict the decision values of faulty processes: a faulty process is allowed to decide differently from non faulty processes even if it fails a very long time after making a decision. Such disagreements may be undesirable since faulty processes may reach inconsistent states and subsequently contaminate the whole system [16]. This is why in the atomic commitment of a distributed database [3] where inconsistent decisions lead the database itself to become inconsistent—which is clearly unacceptable—, one considers a strengthening of the agreement condition, called the *uniform agreement* condition, which precludes any disagreement even due to faulty processes. More formally, the uniform agreement condition specifies that no two processes (whether faulty or not) decide differently [18,19]. The problem that results from substituting uniform agreement for agreement in the consensus specification is called the *uniform consensus* problem.

Consensus originated from a problem in real-time process control (cf. [30,32]). In this context, process decisions are used to trigger some specific actions which must be carried out within strict deadlines. The decisions of faulty processes are ignored in the hope that enough non faulty processes will give their common decisions, so that using their decisions alone, the action will be correctly carried out. The agreement condition is therefore strong enough for such distributed applications, in which the processes that have already decided cannot initiate irreversible actions on their own. This explains why agreement and uniform agreement are relevant safety conditions, according to the type of applications.

No matter what the synchrony of the system is, the uniform agreement condition is trivially not achievable if processes may commit byzantine failures since this failure model imposes no limitation on the possible behaviors, and consequently on the possible decisions of faulty processes. On the other hand, in a synchronous system with n processes, consensus is solvable in the presence of t byzantine failures if $n > 3t$ [25,30]. Dwork et al. [11] showed that non-uniform agreement can be reached for crash, omission, and byzantine failures, for the very realistic partially synchrony models, in which bounds on relative process speeds and on message transmission times exist but are not known or/and hold only after some unknown time. For uniform consensus, things are quite different: since a slow process cannot be distinguished from a crashed one in a non-synchronous system, ensuring agreement with slow processes implies to ensure agreement with the crashed processes. In other words, any algorithm that solves consensus also solves uniform consensus for the crash failure model. Guerraoui [17] used this argument to show that in many partially synchronous systems defined in terms of unreliable failure detectors [4], any algorithm that solves consensus also solves uniform consensus; the argument actually applies to any partially synchronous model defined in [11]. In non-synchronous systems (i.e., both asynchronous and partially synchronous systems) with crash failures, there is thereby no harm to concentrate on consensus instead of uniform consensus. On the other hand, some algorithms that solve consensus in synchronous systems may violate the uniform agreement condition.

So it is interesting to investigate the differences between consensus and uniform consensus in the context of synchronous systems, and the differences in requirements for their solutions depending on the failure model. For byzantine and omission failures, these differences follow from classical results: in the byzantine failure model, consensus is solvable if less than one third of processes are faulty [30]. As mentioned above, uniform consensus is trivially not solvable in systems with byzantine failures no matter how many processes are faulty, and so is harder than consensus. In the omission failure model, the comparison between the two problems is far less immediate. Perry and Toueg [31] exhibited consensus algorithms that tolerate any number of faulty processes. For uniform consensus, we can use the translation given in [29], which transforms any algorithm tolerant of crash failures into one tolerant of omission failures. The translation works only if a minority of processes may fail. As long as this assumption holds, any algorithm that solves uniform consensus in the crash failure model is converted by means of this translation into an algorithm that solves uniform consensus and tolerates omission failures. In systems where half or more processes may fail, Neiger and Toueg [29] showed that uniform consensus cannot be solved with omission failures. As for the byzantine failure model, uniform consensus is therefore harder than consensus for the omission failure model, because its solvability requires more restrictive conditions than consensus.

Our results in this paper concern the crash failure model. Both consensus and uniform consensus are solvable in this model, no matter how many processes are faulty. We show that uniform consensus is still harder than consensus by considering the time complexities of these two problems. For that, we use the well-known synchronized round model of computation, which can be emulated in any synchronous system. In the presence of up to t crash failures, uniform consensus as well as consensus can be solved within $t + 1$ rounds. Moreover, Merritt [27] showed that $t + 1$ is a lower bound on the number of rounds required for deciding in the worst case for both of these problems (see Chapter 6 in [26] and Section 3 *infra* for more detailed references concerning this result). Following [9], we refine this analysis by discriminating runs according to the number of failures f that actually occur. We prove that uniform consensus requires at least $f + 2$ rounds whereas consensus requires only $f + 1$ rounds if f is less than $t - 1$, and both consensus and uniform consensus only require $f + 1$ rounds if $f = t - 1$ or $f = t$.

As a matter of fact, our proof of the lower bound for early deciding in uniform consensus still works when considering a weaker version of uniform consensus introduced by Lamport in [21], which we call *weak uniform consensus*. This latter problem is similar to uniform consensus, except that it requires v to be the only possible decision value only if all the initial values are equal to v *and* there is no failure. Our lower bound thereby holds for the weak uniform consensus problem, and so for any stronger problem. In particular, it holds for the well-known *non-blocking atomic commitment* problem in database systems.

Merritt's result [27] is actually stronger than the one described above since Merritt established the $t + 1$ lower bound for the restricted failure model of "*orderly crash failures*" in which faulty processes must respect the order specified by the protocol in sending messages to neighbors. Therefore, the lower bound for early deciding consensus that we deduce from Merritt's lower bound also holds for orderly crash failures. On the other hand, our proof of the lower bound for early deciding uniform consensus works only for (unordered) crash failures, and we do not know whether this result still holds for the restricted class of orderly crash failures.

We also present consensus and uniform consensus algorithms that achieve our lower bounds for early deciding. In such early deciding algorithms for consensus, processes decide one round earlier than in any uniform consensus algorithm for most cases ($0 \leqslant f \leqslant t - 2$). By refining time complexity analysis as in [9], we thus show that uniform consensus is harder than consensus for the crash failure model.

The lower bound presented here is very close to the one established by Dolev, Reischuk, and Strong [9]: they prove that *consensus* requires at least $\min(f + 2, t + 1)$ rounds before all correct processes can *halt*, i.e., cease executing the algorithm. As already pointed out in [9], it is important to notice the difference between the time at which a process can decide and the time at which it can halt. From the worst case lower bound, we prove that consensus requires $f + 1$ rounds to decide; hence by the lower bound in [9], correct processes cannot stop just after making a decision, in an early deciding algorithm. In turn, our lower bound result implies that, in any early stopping algorithm, processes must postpone deciding to the very end of the computation in order to guarantee agreement uniformity.

Obviously, a lower bound for deciding is also a lower bound for stopping. On the other hand, a lower bound for consensus also holds for its (stronger) uniform version, namely uniform consensus. Consequently, the lower bound results presented in [9] and here are incomparable *a priori*. However, a simple reduction argument (cf. Section 4.2) shows that any lower bound for early deciding uniform consensus is also a lower bound for early stopping consensus. We can thus deduce the lower bound in [9] from ours, except in the case $f = t - 1$ for which Dolev, Reischuk and Strong establish a better result. As our lower bound is actually optimal and because of this particular case $f = t - 1$, there cannot be a simple converse reduction which would allow us to deduce our lower bound from the one in [9].

From a technical viewpoint, our lower bound proof is inspired by the one by Dolev, Reischuk, and Strong, and also proceeds by a double induction. Afterwards, an alternative proof has been given by Keidar and Rajsbaum [20], which uses a single induction but relies on the formalism of *layering* developed by Moses and Rajsbaum [28]. Note that for failure free runs, Lamport [22] also gives the two rounds lower bound, and in [23] he refines the analysis by attaching specific roles to each process (proposer, acceptor, learner).

The paper is organized as follows. Section 2 contains the basic definitions and the formal description of the synchronized round model of computation. Section 3 gives the number of rounds required for deciding in runs of a consensus algorithm with at most f crash failures. We investigate the same question for the uniform consensus problem in Section 4, and in Section 5, we prove that these lower bounds are achievable. Section 6 provides some concluding remarks.

2. The model

We consider synchronous distributed systems consisting of a set of n processes $\Pi = \{p_1, \dots, p_n\}$. Processes communicate by exchanging messages. Communications are point to point. Every pair of processes is connected by a reliable channel. In such systems, one can emulate a computational model called *synchronous model* in which computations are organized in rounds of information exchange. On each process, a round consists of message sending, message receipt, and local processing. We now recall the formal description of the synchronous model (see Chapter 2 in [26] for a detailed presentation): each process p_i has a buffer denoted buffer_i that represents the set of messages that have been sent to p_i but that are not yet received. An algorithm A consists for each process $p_i \in \Pi$ of the following components: a set of states denoted by states_i, an initial state init_i, a message-generation function msgs_i mapping $\mathit{states}_i \times \Pi$ to a unique (possibly *null*) message, and a state transition function trans_i mapping states_i and vectors (indexed by Π) of messages to states_i. In any execution of A, each process p_i, in lock-step, repeatedly performs the following two steps:

(1) Apply msgs_i to the current state to generate the messages to be sent to each process. Put these messages in the appropriate buffers.
(2) Apply trans_i to the current state and the messages present in buffer_i to obtain the new state. Remove all messages from buffer_i.

The combination of these two steps is called a *round* of A. Note that in this model, an algorithm specifies the *set* of messages processes have to send in each round, but not the order in which messages are sent.

We distinguish some of the process states as *halting states*: they are those from which no further activity can occur. When reaching a halting state of algorithm A, a process stops participating to A. That is, from a halting state no messages are sent and the only state transition is a self-loop.

A *run of* A is an infinite sequence of A's rounds. A *partial run of* A is a finite prefix of a run of A.

2.1. Failures

Processes can fail by crashing, that is by stopping in the middle of their executions. A process may crash before or during some instance of the steps described above. A process may thus succeed in sending only a subset of the messages specified to be sent. This can be *any* subset since in this model, a process does not produce its messages sequentially. After crashing at a round, a process does not send any message in any of the subsequent rounds.

A process is said to be *correct* in a run if it does not crash; otherwise it is said to be *faulty*. The set of all the runs of an algorithm A in which at most t processes crash is denoted by $\mathit{Run}(A, t)$.

2.2. Consensus and uniform consensus

In the consensus problem, each process starts with an input value from a fixed value set V and must reach an irrevocable decision on one value of V. The *consensus* specification is defined as the set of all the runs that satisfy the following conditions:

Validity: If all processes start with the same initial value v, then v is the only possible decision value.

Agreement: No two correct processes decide on different values.
Termination: All correct processes eventually decide.

As explained in Section 1, this specification allows processes to decide differently if one of them fails. To avoid such disagreements, the agreement property can be strengthened to

Uniform agreement: No two processes (whether correct or faulty) decide on different values.

The specification that results from replacing agreement by uniform agreement in the consensus specification is called *uniform consensus*. We say that an algorithm *A tolerates t crashes and solves* (*uniform*) *consensus* if all the runs in $Run(A, t)$ satisfy the validity, termination, and (uniform) agreement conditions.

3. Lower bounds for consensus

In this section, we concentrate on the consensus problem and we first recall some well-known lower bound results, namely the lower bound in the worst case [27] and the one for early stopping [9]. We then recall a standard consensus algorithm, originally described in [24], in which all processes that ever decide have decided by the end of round $f + 1$ in all the runs with at most f crash failures. Finally, from the worst case lower bound, we easily deduce that $f + 1$ is indeed a lower bound for deciding in consensus algorithms.

3.1. Lower bound in the worst case

A fundamental result about consensus in synchronous systems is that if $n \geqslant t + 2$, then any consensus algorithm that tolerates t failures must run $t + 1$ rounds in some execution before all processes that ever decide have decided. This lower bound has been originally established for consensus in the case of byzantine failures by Fischer and Lynch [13]. The result was extended first to the case of byzantine failures with authentication by Dolev and Strong [10] and by DeMillo, Lynch, and Merritt [8], and then to crash failures by Merritt [27].[1] Alternative proofs of this worst case lower bound based on bivalency arguments have been then given by Aguilera and Toueg [1], Gafni [15], and Moses and Rajsbaum [28]. Clearly, this worst case lower bound also holds for the stronger problem of uniform consensus. Moreover, it is well-known that there are algorithms for uniform consensus (and so for consensus) tolerating t crash failures, and in which processes decide in $t + 1$ rounds. The lower bound of $t + 1$ rounds is thereby tight for both consensus and

[1] The lower bounds in [10,27] have been actually established for the *byzantine agreement* problem (also called *terminating reliable broadcast*), but can be easily adapted to the consensus problem. As mentioned in Section 1, Merritt [27] proved the $t + 1$ lower bound for the restricted class of "*orderly crash failures*" in which faulty processes must respect the order specified by the protocol in sending messages to neighbors. Merritt's lower bound *a fortiori* holds for our model of (unordered) crash failures.

uniform consensus. With respect to the worst case time complexity, consensus and uniform consensus are therefore two equivalent problems.

3.2. Lower bound for early stopping

Following [9], we refine the analysis by discriminating runs according to the number of failures that *actually* occur: we consider the number of rounds required to decide not over all the runs of an algorithm that tolerates t crash failures, but over all the runs of the algorithm in which at most f processes crash for any f, $0 \leqslant f \leqslant t$. For consensus algorithms, Dolev, Reischuk, and Strong [9] give a lower bound on the number of rounds required for processes *to halt* in the runs with at most f faulty processes. More precisely, they prove the following theorem:

Theorem 3.1 (Dolev et al., 1990). *Let A be a consensus algorithm that tolerates t process crashes. If $n \geqslant t+2$ then for each f, $0 \leqslant f \leqslant t$, there exists a run of A with at most f crash failures such that some process has not halted before round* $\min(t+1, f+2)$ *in that run.*

As pointed out by Dolev, Reischuk, and Strong in [9], it is important to notice that a process may decide at some round without reaching a halting state, namely, it may continue to send messages and to participate to the consensus algorithm in subsequent rounds. In other words, there may be a difference between the time at which a process decides and the time at which it halts. Indeed, Theorem 3.1 gives the number of rounds until the processes all stop but says nothing about the time when processes decide. Note that obviously, a lower bound on deciding is also a lower bound on halting, but not *vice-versa*.

In this paper and contrary to [9], we consider the time at which processes decide and not the time at which they halt. This is motivated by the following reasons. Firstly, from a practical viewpoint, the time at which decisions are taken is a significant time measure: it is indeed quite important to determine the time when decisions are available in the system. Secondly, since the $t+1$ worst case lower bound result considers the decision time and not the halting time, it seems more relevant to keep the same time complexity measure when refining efficiency analysis of consensus and uniform consensus algorithms.

3.3. An early deciding algorithm

We now present a well-known early deciding consensus algorithm devised by Lamport and Fischer [24] which will prove that the lower bound of Theorem 3.1 does not hold when “early stopping” is replaced by “early deciding”.

In the algorithm which we call *EDAC*, each process p_i maintains a variable *Failed* containing the set of processes that p_i detects to have crashed. Process p_i learns that p_j crashes during a round if p_i receives no message from p_j at this round. At the end of every round, each process p_i updates its variable *Failed*. If *Failed* remains unchanged during round r, that is if p_i detects no new crash failure, and if p_i has not yet decided, then p_i decides at the end of round r. Any process that decides on v at round r broadcasts a (D, v) message at round $r+1$ to inform the other processes of its decision and to force the

```
states_i
    rounds ∈ N, initially 0
    $W \subseteq V$, initially the singleton set consisting of $p_i$'s initial value
    done, a Boolean, initially false
    halt, a Boolean, initially false
    $Rec\_Failed \subseteq \Pi$, initially ∅
    $Failed \subseteq \Pi$, initially ∅
    $decision \in V \cup \{unknown\}$, initially unknown

msgs_i
    if ¬halt then
        if ¬done then send $W$ to all processes
        else send $(D, decision)$ to all processes

trans_i
    if ¬halt then
        $rounds := rounds + 1$
        let $X_j$ be the message from $p_j$, for each $p_j$ from which a message arrives
        if done then halt := true
        if some message $(D, v)$ arrives then
            $decision := v$
            done := true
        else $W := W \cup \bigcup_j X_j$
            $Rec\_Failed := Failed$
            $Failed := \{p_j$: no message arrives from $p_j$ at the current round$\}$
            if $Rec\_Failed = Failed$ then
                $decision := \min(W)$
                done := true
```

Fig. 1. The *EDAC* algorithm.

processes that have not yet decided to decide on v in turn. The code of *EDAC* is given in Fig. 1 (in this code null messages do not appear in the $msgs_i$'s).

Among $f + 1$ rounds of a run with at most f faulty processes, there must be some round at which no process fails. So each process p definitely detects a failure free round (maybe erroneously), and at the end of such a round, p knows all the initial values in play at this time. The other processes cannot learn any other initial values; hence it is safe for p to decide at the end of the first round at which it has detected no new failure. The reader is referred to [24] for a complete correctness proof of *EDAC*.

The *EDAC* algorithm proves that the $f + 2$ lower bound of Theorem 3.1 does not hold when considering the question of early deciding instead of the one of early stopping. This shows that there is an actual difference between the time at which a process can decide and the time at which it can halt (this observation has been already mentioned by Dolev, Reischuk, and Strong in [9]). As exemplified by the *EDAC* algorithm, it is not safe for a process to stop just after making a decision: it may be the case that some process p_i receives a new information from process q at some round r in which p_i detects no new failure and this information affects p_i's decision value. Since some process p_j may receive no message from q (because q fails) at round r, the only way p_i is certain that the information gets at p_j is that p_i itself sends it. Thus, process p_i has to send this information to p_j, and so cannot stop as soon as it makes a decision.

3.4. A lower bound for early deciding

We now prove that the *EDAC* algorithm is optimal, i.e., at least $f+1$ rounds are required for deciding in some run with at most f faulty processes. This result is a straightforward consequence of the $t+1$ worst case lower bound.

Theorem 3.2. *Let A be a consensus algorithm that tolerates t process crashes. If $n \geqslant t+2$ then for each f, $0 \leqslant f \leqslant t$, there exists a run of A with at most f crashes in which at least one process decides not earlier than during round $f+1$.*

Proof. Let f be any fixed element in $\{0, \dots, t\}$. The worst case lower bound [27] recalled in Section 3.1 applied to the algorithm A, seen as a consensus algorithm tolerating f crash failures, shows that there exists a run of A with at most f crashes in which some correct process decides not earlier than during round $f+1$. $\square$

Compared with the lower bounds stated in Theorem 3.1, this result shows that for most of the cases ($0 \leqslant f \leqslant t-2$), it takes at least one additional round to halt after making a decision in early deciding algorithms for consensus.

4. A lower bound for early deciding uniform consensus

We now study the question of early deciding for uniform consensus. As the uniform consensus specification is stronger than the one of consensus, the $f+1$ lower bound stated in Theorem 3.2 *a fortiori* holds for uniform consensus. In this section, we refine this result by proving a lower bound for uniform consensus greater than $f+1$. Since the $t+1$ lower bound for consensus is achievable (for example, by the *EDAC* algorithm), this thereby shows that the uniform consensus problem is harder than the consensus problem in the context of synchronous model with crash failures. We then show how to deduce the lower bound for early stopping consensus established by Dolev, Reischuk, and Strong (Theorem 3.1) from our lower bound for early deciding uniform consensus, except in the case $f = t-1$. Finally, we observe that our lower bound result also applies to other agreement problems with a uniform agreement property, and in particular to the non-blocking atomic commitment problem.

4.1. A lower bound for early deciding uniform consensus

Theorem 4.1. *Suppose $t \geqslant 1$ and let A be a uniform consensus algorithm that tolerates t process crashes. For each f, $0 \leqslant f \leqslant t$, there exists a run of A with at most f crashes in which at least one process decides not earlier than during round $f+2$ if $f \leqslant t-2$, and not earlier than during round $f+1$ otherwise.*

Here we use the same proof technique as in [9]. However, contrary to the computational model in [9], there is here no special halting states, and so a process which has stopped to send messages has necessarily crashed. This makes the proof simpler at various points, but requires to have more processes that can be faulty in the construction of some specific runs. This latter point has an actual impact when $f = t - 1$: in this case, our lower bound for early deciding is smaller than the one in [9] for early stopping. All these differences lead us to present the complete proof of Theorem 4.1 even if it is inspired by the one given in [9].

Before proving the theorem, we first introduce some additional definitions and notation. Let ρ be a run of an algorithm A. For any $k \leqslant 1$, we define $\rho[k]$ to be the partial run of A that consists of the k first rounds of ρ. The *conservative extension* of $\rho[k]$ is the unique run ρ' of A such that $\rho'[k] = \rho[k]$ and no process crashes after round k. We say that ρ is *f-regular* if there are at most f processes that crash in ρ and for every k, $1 \leqslant k \leqslant f$, there are at most k processes that crash in $\rho[k]$. If a process crashes at round k and fails to send message m, and if no process crashes after round k, then m is said to be a *last unsent message in ρ*.

For any initial configuration C, there exists a unique failure free run of A that starts from C; this run is denoted by r_C. On the other hand, for any process p, A admits a unique run ρ_C^p starting from C and in which only p is faulty and crashes from the beginning; this run is called *the silencing of p from C*.

If ρ and ρ' are two runs of A, we say that ρ is *indistinguishable from ρ' with respect to process p*, denoted $\rho \sim_p \rho'$, if p has the same initial state and performs the same sequence of actions in ρ and ρ'.

Finally, if A solves the uniform consensus problem, then for any run ρ of A, *dec*(ρ) denotes the unique value that is decided in ρ.

Proof of Theorem 4.1. Since our concern is a lower bound result, we can restrict our attention to the binary uniform consensus problem, i.e., $V = \{0, 1\}$.

First, in the case $t = 1$, the lower bound in Theorem 4.1 is captured by Theorem 3.2, and so is already proved.

Thus we now assume that $t \geqslant 2$. Let A be any algorithm that solves the uniform consensus problem and that tolerates t crashes. For any integer f, $0 \leqslant f \leqslant t$, consider the set of all runs of A in which at most f processes crash. There are three cases to consider.

(1) $f = 0$. We give a straightforward proof in this case. Assume for the sake of contradiction that in every failure free run of A all the processes decide at the first round. Let C^0 and C^n denote the initial configurations such that all processes have initial value 0 and 1, respectively. Consider a chain of initial configurations $C^0, C^1, \ldots, C^n$ spanning C^0 to C^n such that any two consecutive configurations C^{i-1} and C^i differ only in the initial value of process p_i. Let r_i denote the failure free run of A starting from C^i, i.e., $r_i = r_{C^i}$. We now prove by induction on i that the value decided in each r_i is 0. By validity, all processes must decide 0 in r_0. Let i be such that $1 \leqslant i \leqslant n$, and suppose that the decision value in r_{i-1} is 0. Let p be a process different to p_i; consider the run σ_{i-1} starting from C^{i-1} such that (1) all the processes are correct except processes p and p_i, (2) p_i succeeds in sending only one message to p and then crashes, and (3) p crashes just at the beginning of the second round. Since $t \geqslant 2$ and A tolerates t crashes, σ_{i-1} is a run of *Run*(A, t), and so satisfies the three conditions of uniform consensus. The first rounds $\sigma_{i-1}[1]$ and $r_{i-1}[1]$ are indistinguishable to process p. By inductive hypothesis, p decides 0 in $r_{i-1}[1]$, and so in $\sigma_{i-1}[1]$. Now, we consider the run τ_i that is identical to σ_{i-1} except that τ_i starts from C^i

(instead of C^{i-1}). Since $n \geqslant 3$, there exists some process q such that $q \neq p$ and $q \neq p_i$. For such a process q, we have $\tau_i \sim_q \sigma_{i-1}$. This implies that $dec(\tau_i) = dec(\sigma_{i-1}) = 0$. The first rounds $\tau_i[1]$ and $r_i[1]$ are indistinguishable to p, and so p decides 0 in r_i. So $dec(r_i) = 0$, as needed. In particular, $dec(r_n) = 0$. This contradicts that the decision value in r_n must be 1 because of the validity condition.

(2) $f \in \{1, \ldots, t-2\}$. First, we use a bivalency argument borrowed from [14] to show that there is an initial configuration C from which the failure free run and the silencing of some process lead to two different decision values (Lemma 4.2). We then proceed by contradiction: we first show that if in all the runs of A with at most f crashes, all processes decide by the end of round $f+1$, then any last unsent message of a f-regular run can be "added" without altering the decision value (Lemma 4.3). By successive application of this intermediate result, we obtain that all the f-regular runs starting from some initial configuration C lead to the same decision value as the failure free run r_C. In particular, any silencing of some process from C has the same decision value as r_C, contradicting the preliminary bivalent result (Lemma 4.2).

(3) $f = t-1$. The case studied above provides a run with at most $t-2$ crashes (and so with at most $t-1$ crashes), in which uniform consensus is achieved not earlier than at round $f+1 = t$.

(4) $f = t$. In this case, the lower bound immediately follows from the $t+1$ worst case lower bound. □

Lemma 4.2. *There is an initial configuration C and there is some process p such that $dec(r_C) \neq dec(\rho_C^p)$.*

Proof. By the standard bivalency argument of [14], there are two initial configurations C and C' which differ only by the initial value of some process p and such that (1) $dec(r_C) \neq dec(r_{C'})$. Clearly, for any process $q \neq p$, we have $\rho_C^p \sim_q \rho_{C'}^p$, and thus (2) $dec(\rho_C^p) = dec(\rho_{C'}^p)$. From (1) and (2), it follows that $dec(r_C) \neq dec(\rho_C^p)$ or $dec(r_{C'}) \neq dec(\rho_{C'}^p)$. □

Lemma 4.3. *Let f be an integer, $0 \leqslant f \leqslant t-2$. Suppose that in all the runs of A with at most f crashes, all the processes that are still alive at the end of round $f+1$ have decided by the end of round $f+1$. Let σ be an f-regular run of A and let m be any last unsent message of σ. If τ is the run of A which is identical to σ except that m is sent in τ, then $dec(\sigma) = dec(\tau)$.*

Proof. By definition of τ and since σ is f-regular, τ is also an f-regular run of A. Thus, after $f+1$ rounds, all the processes that are still alive have decided in both σ and τ. Note that any process is correct in τ iff it is correct in σ.

Let p be the process that fails to send m and q be the destination of m. Let k be the round of σ during which p crashes. The cases where $k > f+1$ are trivial. Thus, we assume that $k \leqslant f+1$. The proof is by decreasing induction on k, starting with $k = f+1$ and ending with $k = 1$. It will be convenient to note l the complement to k in $f+1$, i.e., $l = f+1-k$.

Basis: $k = f+1$. Since $n - f \geqslant n - (t-2) \geqslant 3$, there exists at least one process, different from q, that is correct in both σ and τ. For such a process s, we have

$\sigma[f+1] \sim_s \tau[f+1]$. This implies that s decides the same value in σ and τ. Therefore, $dec(\sigma) = dec(\tau)$.

Inductive step: Assume $k \geqslant f$. Suppose the claim is true for any last unsent message in round i of any f-regular run, with $k+1 \leqslant i \leqslant f+1$. Run σ is f-regular, and so there are at most k processes that crash in $\sigma[k]$. Since $k+1+l = f+2$ and $f+2 < n$, we can find l processes $r_1, \ldots, r_l$ which are different from q, and which do not crash in $\sigma[k]$. For convenience, we note $q = r_0$. Let σ' be the run that is identical to σ, except that:

- At round $k+1$, r_0 succeeds in sending a message only to r_1 and then crashes. No other processes fail in this round.
- At round $k+2$, r_1 succeeds in sending a message only to r_2 and then crashes. No other processes fail in this round.

  $\vdots$
- At round $f+1$, r_{l-1} succeeds in sending a message only to r_l and then crashes.
- Process r_l crashes at the beginning of round $f+2$, just before sending any message. No other processes fail in this round and in the later rounds.

Run σ is regular, and thus there are at most $k+l+1 = f+2$ crash failures in σ'. Since $f \leqslant t-2$, σ' is in $Run(A,t)$.

Let $\sigma^1, \ldots, \sigma^l$ denote the conservative extensions of $\sigma'[k+1], \ldots, \sigma'[f+1]$, respectively. We can safely extend this notation to $\sigma^0 = \sigma$ because m is a last unsent message of σ. Since σ is regular, there are at most $k+i$ crash failures in σ^i. In particular, there are at most f crash failures in σ^{l-1}. Process r_l is correct in σ^{l-1}, and thus decides by the end of round $f+1$ in σ^{l-1}. Moreover, σ', σ^{l-1}, and σ^l are indistinguishable to r_l up to the end of round $f+1$. This shows that process r_l decides the same value by the end of round $f+1$ in each of these three runs. Since the agreement property is uniform, this implies that

$$dec(\sigma^{l-1}) = dec(\sigma^l) = dec(\sigma'). \tag{1}$$

On the other hand, in each run σ^i, $1 \leqslant i \leqslant l$, the message that r_{i-1} fails to send to any process $s \notin \{p, r_0, r_1, \ldots, r_i\}$ at round $k+i$ is a last unsent message of σ^i. Moreover, $\sigma^0, \sigma^1, \ldots, \sigma^{l-1}$ are f-regular runs. By successive application of the inductive hypothesis, we obtain that $dec(\sigma^{i-1}) = dec(\sigma^i)$ for any index i such that $1 \leqslant i \leqslant l-1$. Finally, this shows that

$$dec(\sigma^{l-1}) = \cdots = dec(\sigma^1) = dec(\sigma^0). \tag{2}$$

Equalities (1) and (2) imply that $dec(\sigma) = dec(\sigma')$.

Now from run τ, we use a similar construction of regular runs: let τ', and $\tau^0 = \tau, \tau^1, \ldots, \tau^l$ denote the so-defined regular runs of A. By a similar argument to those used with σ', σ^{l-1}, and σ^l, we show that

$$dec(\tau^{l-1}) = dec(\tau^l) = dec(\tau'). \tag{3}$$

By repeated applications of the inductive hypothesis, we get that

$$dec(\tau^{l-1}) = \cdots = dec(\tau^1) = dec(\tau). \tag{4}$$

This implies that $dec(\tau) = dec(\tau')$.

On the other hand, let s be a process that is correct in both σ' and τ' (such a process exists since $f + 2 \leqslant t < n$). Runs σ' and τ' are indistinguishable to s, i.e., $\sigma' \sim_s \tau'$. This implies that $dec(\sigma') = dec(\tau')$. So $dec(\sigma) = dec(\tau)$, as needed. □

For any $t \geqslant 1$ and any f, $0 \leqslant f \leqslant t$, the lower bound for early deciding uniform consensus in Theorem 4.1 is equal to the one for early stopping consensus in Theorem 3.1, except the case $f = t - 1$. In this case, we have only proved that t rounds are necessary to decide in a uniform consensus algorithm (as well as in consensus algorithms) while $t + 1$ rounds are required before halting. Now the important point is to determine whether our lower bound is optimal. If so, making a uniform decision and halting require the same number of rounds, except when $f = t - 1$, in which case the (uniform) decision can be taken one round earlier.

4.2. A lower bound for early stopping consensus

As noticed in Section 3, a lower bound for deciding is also a lower bound for stopping, while a lower bound for consensus is also a lower bound for uniform consensus. Hence, the two lower bounds in Theorems 3.1 and 4.1 are *a priori* incomparable. However, we are going to prove that the problem of deciding in uniform consensus is indeed reducible to the one of stopping in consensus.[2]

For that, consider a consensus algorithm A in which each correct process eventually reaches a halting state; A can be transformed into an algorithm $B = T(A)$ which is identical to A, except that each process postpones its decision until it halts (the decision value in B is thus the same as in A).

Proposition 4.4. *The algorithm $B = T(A)$ solves the uniform consensus problem.*

Proof. By definition of $B = T(A)$, any run ρ of B derives from the run σ of A identical to ρ except that a process makes a decision in ρ at the time it stops in σ. Clearly, the validity and termination conditions are carried over from σ to ρ. For uniform agreement, suppose that processes p and q decide v and v' at rounds r and r' in run ρ, respectively. This means that in σ, p and q also decide v and v' and halt at rounds r and r'. It may be the case that p (or q) crashes in run σ; then the failure occurs only after round r (or r'), and so has no impact. Consequently, there is a run σ' of A in which p and q are correct and decide v and v', respectively. Since σ' satisfies agreement, we have $v = v'$ as needed. □

Early deciding uniform consensus is therefore reducible to early stopping consensus. Since the reduction takes no additional round, we obtain the following corollary:

Corollary 4.5. *A lower bound for early deciding uniform consensus also holds for early stopping consensus.*

[2] This result has been inspired by a suggestion of one reviewer of the first version of this paper.

This corollary combined with Theorem 4.1 provides a lower bound for early stopping consensus similar to the one of Theorem 3.1 except the case $f = t - 1$ for which the lower bound given by Dolev, Reischuk, and Strong is better.

4.3. A general lower bound for early deciding

Interestingly, the proof of Theorem 4.1 only uses the weaker version of validity condition introduced by Lamport [21], which is:

Weak validity: If all processes are correct and start with the same initial value v, then v is the only possible decision value.

Consequently, the lower bound in Theorem 4.1 still holds for *weak uniform consensus* (the problem defined by the termination, weak validity, and uniform agreement conditions), and so for *non-blocking atomic commitment* since the specification of this latter problem is stronger than the one of weak uniform consensus, as noticed by Hadzilacos [18]. However, the lower bound in Theorem 4.1 and the worst case lower bound $t + 1$ as well, do not hold anymore when considering the very weak validity condition in [14] that only stipulates that there are at least two possible decision values. Indeed, Dwork and Moses [12] devised a two rounds synchronous algorithm, which solves this very weak agreement problem. Coming back to the proof of Theorem 4.1, we observe that this is due to Lemma 4.2 which is no more true for this latter agreement problem.

5. An early deciding algorithm for uniform consensus

In this section, we show that the lower bound in Theorem 4.1 is tight. For that, we might think just to apply the reduction in Section 4.2 to the optimal algorithm described in [9] that achieves the lower bound in Theorem 3.1. Unfortunately, this algorithm, which is very robust in the sense that it tolerates byzantine failures, is proved to work only when $n > \max(4t, 2t^2 - 2t + 2)$. Moreover, by this reduction-based method, the resulting algorithms for early deciding uniform consensus do not achieve our lower bound in the case $f = t - 1$, which indicates this is the more difficult case to handle.

We start by considering the particular case $t = 1$. If $t \geqslant 2$, then we prove that for any f, $0 \leqslant f \leqslant t$, there exists an algorithm for uniform consensus that achieves the lower bound in Theorem 4.1. Finally, we show that all the algorithms for the different values of f can be combined to yield a single algorithm that achieves our lower bound alone.

5.1. An optimal 1-resilient algorithm

In the case $t = 1$, Charron-Bost et al. [6] describe a two rounds uniform consensus algorithm tolerating one crash failure in which processes decide at the end of the first round in a failure free run. This algorithm, called *TwoCoord*, is based on the following ideas: the first round is coordinated by process p_1 which broadcasts its initial value v_1. Upon receiving v_1, any process p_i decides v_1 at the end of round 1 and reports its decision at round 2. If p_2 has received no message from p_1 in the first round (because p_1 has crashed), p_2 coordinates round 2 and broadcasts its initial value v_2. Since at most one failure may occur, every correct process has received v_1 or v_2, or both by the end of the second round. The decision value v_1 prevails, that is if a process receives v_1, then it decides v_1; otherwise it decides v_2.

The correctness of *TwoCoord* relies on the fact that if p_1 succeeds in sending v_1 in the first round to some process p, $p \neq p_1$, and if $t = 1$, then p or p_1 is correct, and so v_1 can be definitely delivered to all processes by the end of the second round. The $f + 1$ lower bound established for early deciding consensus in the case $t = 1$ is thereby tight.

5.2. The EDAUC and Tree$_t$ algorithms

We now suppose that $t \geqslant 2$. The *EDAC* algorithm presented in Section 3.3 does not solve the uniform consensus problem. To see that, consider a run of *EDAC* in which all processes are correct, except p_i and p_j, and all the initial values equal 1, except p_j's initial value that is equal to 0. Suppose p_j crashes at the first round and succeeds in sending a message only to p_i, whereas p_i crashes at the very beginning of round 2. Process p_i cannot detect p_j's crash, and so decides on 0 at the end of the first round just before crashing. The other processes make a decision at round 3; since they never receive p_j's initial value, they decide on 1. Note that this is the same reason why p_i cannot stop just after making a decision without risking the violation of the agreement property. This again points out that the questions of early stopping consensus and early deciding uniform consensus are closely related.

However, it is easy to design a variant that solves uniform consensus. For that, we adapt the *EDAC* algorithm by postponing decision after broadcasting the decision value to all at the next round. This variant, called *EDAUC*, clearly achieves the $f + 2$ lower bound of Theorem 4.1 for every f, $0 \leqslant f \leqslant t - 2$.

The case $f = t - 1$ is more tricky. To handle this case, we introduce a uniform consensus algorithm *Tree$_t$* in $t + 1$ rounds, that tolerates t crash failures and such that processes have all decided by the end of round t if there are less than t faulty processes. This proves that for any f, $0 \leqslant f \leqslant t$, the lower bound in Theorem 4.1 is tight.

The *Tree$_t$* algorithm is actually a generalization for an arbitrary value of t of the *TwoCoord* algorithm. It is based on the following idea: Processes $p_1, \ldots, p_{t+1}$ broadcast their initial values during the first round. Process p_j decides v_1 (p_1's initial value) if it knows that p_1 has succeeded in sending v_1 to all the processes in the first round. In general, p_j decides v_i (p_i's initial value) if p_j can decide neither v_1 nor $v_2, \ldots,$ nor v_{i-1}, and p_j knows that p_i has sent its initial value to all the processes in the first round. Since at most t processes may crash, each process eventually decides some value of $\{v_1, \ldots, v_{t+1}\}$. If process p_j receives a message from p_i in the second round, p_j can safely deduce that p_i has not crashed during the first round and thus p_i has sent a message to all the processes in the first round. If that is not the case, how can p_j know whether p_i has succeeded in sending a message to all the processes in the first round? We claim that p_j needs only t rounds to determine whether p_i has failed or not in sending messages at the first round of a run in which at most $t - 1$ processes crash. For this purpose, we use a strategy known as *exponential information gathering* (*EIG*, for short) introduced in [2]. The basic structure used by *EIG* algorithms is a labelled tree, whose paths from the root represents chains of processes along which some values are propagated.

In the *Tree*$_t$ algorithm, each process maintains t *EIG* trees that are denoted by $T^1, \ldots, T^t$. Each tree T^i has $t+1$ levels, ranging from 0 (the root) to the level t (the leaves). Each node at level k, $0 \leqslant k \leqslant t-1$, has exactly $n-k-1$ children. Each node in T^i is labelled by a string of process indices as follows: the root is labelled by the empty string λ, and each node with label $i_1 \cdots i_k$ has $n-k-1$ children with labels $i_1 \cdots i_k l$ where l ranges over all the elements of $\{1, \ldots, n\} \setminus \{i_1, \ldots, i_k, i\}$. In other words, all the chains of T^i consist of distinct processes that are all different to p_i. In the course of the computation, the processes decorate the nodes of their trees with values in $\{0, 1, \mathit{null}\}$. Nodes at level k are decorated during the round $k+1$. Process p_j decorates the root of T^i by 1 or 0 depending on whether a message from p_i has arrived or not at p_j during the first round. The node labelled by $i_1 \cdots i_k$ in T^i is decorated by p_j with 1 if p_{i_k} has told p_j at round $k+1$ that $p_{i_{k-1}}$ has told p_{i_k} at round k that $\ldots p_{i_1}$ has told p_{i_2} at round 2 that p_{i_1} has received a message from p_i at round 1. On the other hand, $i_1 \cdots i_k$ in T^i is decorated by p_j with 0 means that p_{i_k} has told p_j at round $k+1$ that $p_{i_{k-1}}$ has told p_{i_k} at round k that $\ldots p_{i_1}$ has told p_{i_2} at round 2 that p_{i_1} has not received a message from p_i at round 1. Moreover, if the node labelled by $i_1 \cdots i_k$ in T^i is decorated by *null*, then it means that the chain of communication $p_{i_1}, \ldots, p_{i_k}, p_j$ has been broken by a crash failure.

At round t, if process p_j detects less than t crashes (i.e., p_j receives at least $n-t+1$ messages), then p_j makes a decision; otherwise, p_j decides at round $t+1$. Unless p_j learns that some process has already decided some value v (in which case p_j decides on v), p_j decides on the initial value v_i of p_i if p_j knows that at the first round, (1) $p_1, \ldots, p_{i-1}$ have crashed and (2) p_i has succeeded in sending v_i to all processes. Conditions (1) and (2) are characterized by the fact that 0 occurs in all the trees $T^1, \ldots, T^{i-1}$, and 0 does not occur in T^i.

The formal definition of the *Tree*$_t$ algorithm is given in Fig. 2. In this algorithm, for any index $i \in \{1, \ldots, t\}$ and for every string x that occurs as a label of T^i, each process has a variable $\mathit{val}(x)^i$; the set of values that decorate T^i is denoted by $\mathit{Val}(T^i)$. If $X = \{\mathit{val}(x)^i \colon |x| = k-1,\ i \notin x,\ 1 \leqslant i \leqslant t\}$ arrives from p_j at round k then $\mathit{update}(T^1, \ldots, T^t, X)$ denotes the multiple assignment:

$$\mathit{val}(xj)^i := \mathit{val}(x)^i, \quad 1 \leqslant i \leqslant t,\ |x| = k-1,\ i \notin x,\ j \notin x,\ \text{and } i \neq j.$$

On the other hand, if no message arrives from p_j at round k, then $\mathit{update}(T^1, \ldots, T^t, \mathit{null}^*)$ denotes the multiple assignment:

$$\mathit{val}(xj)^i := \mathit{null}, \quad 1 \leqslant i \leqslant t,\ |x| = k-1,\ i \notin x,\ j \notin x,\ \text{and } i \neq j.$$

In the sequel, we use the subscript j to denote the instance of a state component belonging to process p_j.

To prove that *Tree*$_t$ solves uniform consensus, we first give two lemmas that relate the values of the various T^i. The first lemma describes the relationships between *vals* at different processes at adjacent levels in the trees T^k.

```
states_i
    $rounds \in N$, initially 0; $T^1, \dots, T^t$, whose all values are equal to unknown
    $w^1, \dots, w^{t+1} \in V \cup \{unknown\}$, initially unknown; $v \in V$, initially $p_i$'s initial value
    $decision \in V \cup \{unknown\}$, initially unknown; $done \in \{true, false\}$, initially false

msgs_i
    case
        $round = 0$:
            if $1 \leqslant i \leqslant t+1$ then send $v$ to all processes
            else send null to all processes
        $round = 1, \dots, t-1$:
            send $\{val(x)^j\colon\ |x| = rounds - 1,\ j \notin x,\ 1 \leqslant j \leqslant t\}$ to all processes
        $round = t$:
            if $\neg done$ then send $\{val(x)^j\colon\ |x| = t - 1,\ j \notin x,\ 1 \leqslant j \leqslant t\}$ to all processes
            else send $(D, decision)$ to all processes

trans_i
    $rounds := rounds + 1$
    let $X_j$ be the message from $p_j$, for each $p_j$ from which a message arrives
    case
        $rounds = 1$:
            for all $j \in \{1, \dots, t+1\}$ do
                if a message has arrived from $p_j$ then
                    $w^j := X_j$
                    if $j \neq t+1$ then $val(\lambda)^j := 1$
                else if $j \neq t+1$ then $val(\lambda)^j := 0$
        $rounds = 2, \dots, t$:
            for all $j \in \{1, \dots, n\}$ do
                if a message has arrived from $p_j$ then $update(T^1, \dots, T^t, X_j)$
                else $update(T^1, \dots, T^t, null^*)$
            if $rounds = t$ then
                if at least $n + 1 - t$ messages have arrived then
                    $done := true$
                    if $0 \notin Val(T^1)$ then $decision := w^1$
                    else if $0 \notin Val(T^2)$ then $decision := w^2$
                        else
                            $\ddots$
                               if $0 \notin Val(T^{t-1})$ then $decision := w^{t-1}$
                               else $decision := w^t$
        $rounds = t + 1$:
            if $\neg done$ then
                if some message $X_j$ is equal to $(D, decision_j)$ then $decision := decision_j$
                else if $0 \notin Val(T^1)$ then $decision := w^1$
                        else if $0 \notin Val(T^2)$ then $decision := w^2$
                            else
                                $\ddots$
                                   if $0 \notin Val(T^t)$ then $decision := w^t$
                                   else $decision := w^{t+1}$
```

Fig. 2. The $Tree_t$ algorithm.

Lemma 5.1. *After t rounds of the $Tree_t$ algorithm, for any node label y of T^k such that $val(y)_i^k \neq null$ and for any prefix xj of y, x is a node label of T^k such that $val(x)_j^k = val(y)_i^k$. In particular, $val(x)_j^k = val(xj)_i^k$.*

Proof. Obvious from the definition of the *update* procedure. □

The second lemma describes when 0 occurs in some tree T^k.

Lemma 5.2. *If* 0 *occurs in the set $Val(T^k)$ at any process, then p_k crashes in round* 1.

Proof. Suppose $0 \in Val(T^k)_i$, i.e., there exists a node label x of T^k such that $val(x)_i = 0$. We claim that there is some process index j such that $val(\lambda)_j^k = 0$: if $x = \lambda$ then $j = i$. Otherwise $x = i_1 \cdots i_l$ and Lemma 5.1 implies that $val(\lambda)_{i_1}^k = 0$. In this case, we have $j = i_1$.

From the algorithm, $val(\lambda)_j^k = 0$ if p_k fails in sending its initial value to p_j and thus crashes during the first round. □

The following lemma describes the set of possible decision values.

Lemma 5.3. *The decision value of any process is the initial value of some process in* $\{p_1, p_2, \dots, p_{t+1}\}$.

Proof. Suppose any process p_i decides v in round r. From the algorithm, $r = t$ or $r = t + 1$.

(1) $r = t$. From the code of $Tree_t$, it follows that p_i receives at least $n + 1 - t$ messages in round t and there exists an index $j \in \{1, \dots, t\}$ such that p_i decides the current value of the p_i's variable denoted w_i^j (cf. Fig. 2).

(a) If $1 \leqslant j \leqslant t - 1$ then, from the algorithm, we have $0 \notin Val(T^j)_i$. In particular, $val(\lambda)_i^j = 1$ and w_i^j is assigned to v_j (p_j's initial value) in the first round. This shows that $v = v_j$.
(b) If $j = t$ then $0 \in Val(T^1) \cap \cdots \cap Val(T^{t-1})$. Lemma 5.2 shows that $p_1, \dots, p_{t-1}$ have crashed in the first round. Since p_i has received at least $n + 1 - t$ messages in round t, a message has arrived from p_t in this round and thus p_t may not have crashed during the first round. Therefore, p_i has received p_t's initial value v_t at round 1, and then p_i has set w_i^t to v_t at the end of the first round.

(2) If $r = t + 1$ there are two cases to consider:

(a) Process p_i decides $v = decision_j$ by receiving a message $(D, decision_j)$. From the algorithm, it is clear that p_j has decided at round t. From the above case, it follows that p_j's decision value is in $\{v_1, \dots, v_t\}$. Therefore v also belongs to $\{v_1, \dots, v_t\}$.
(b) Process p_i receives no $(D, decision_j)$ message in round $t + 1$. In this case, p_i decides some w_i^j with $j \in \{1, \dots, t + 1\}$. There are two cases to consider:

(i) $1 \leqslant j \leqslant t$. From the algorithm, we have $0 \notin Val(T^j)$. In particular, $val(\lambda)_i^j = 1$ and w_i^j is set to v_j. Thus, p_i decides $v = v_j$.

(ii) $j = t + 1$. In this case, $0 \in Val(T^1) \cap \cdots \cap Val(T^t)$. From Lemma 5.2, we deduce that $p_1, \ldots, p_t$ have crashed in the first round. Since at most t processes crash, p_{t+1} is correct and has sent its initial value v_{t+1} to p_i in the first round. Therefore, p_i has set its variable w_i^{t+1} to v_{t+1} at round 1, and thus p_i decides v_{t+1}. □

The next two lemmas provide the key arguments to the uniform agreement property.

Lemma 5.4. *If p_i decides v and p_j decides v' both at round t then $v = v'$.*

Proof. The proof is by contradiction. Suppose that in round t, p_i and p_j decide v and v', respectively, and $v \neq v'$. In this case, p_i and p_j receive at least $n + 1 - t$ messages in round t. From Lemma 5.3, there are two indices k and l such that $v = v_k$ and $v' = v_l$. Since $v \neq v'$, we have $k \neq l$. For example, assume that $k < l$. From the algorithm, $0 \notin Val(T^k)_i$ and $0 \in Val(T^k)_j$. Consequently, there exists some node label x in T^k such that

$$val(x)_i^k \neq 0, \qquad val(x)_j^k = 0, \quad \text{and} \quad 0 \leqslant |x| \leqslant t - 1.$$

There are two cases to consider.

(1) $0 \leqslant |x| \leqslant t - 2$. In this case, p_j sends $val(x)_j^k = 0$ to p_i in round $|x| + 2 \leqslant t$ and thus $val(xj)_i^k = 0$. But $0 \notin Val(T^k)_i$—a contradiction.

(2) $|x| = t - 1$, i.e., there are some process indices $i_1, \ldots, i_{t-1}$ such that $x = i_1 \cdots i_{t-1}$, and so

$$val(i_1 \cdots i_{t-1})_j^k = 0 \quad \text{and} \quad val(i_1 \cdots i_{t-1})_i^k \neq 0.$$

From Lemma 5.1, we have:

$$val(\lambda)_{i_1}^k = val(i_1)_{i_2}^k = \cdots = val(i_1 \cdots i_{t-2})_{i_{t-1}}^k = val(i_1 \cdots i_{t-1})_j^k = 0.$$

Moreover, for any non empty prefix y of $i_1 \cdots i_{t-1}$, $val(y)_i^k = null$, otherwise $val(y)_i^k = val(\lambda)_{i_1}^k = 0$—a contradiction with the fact that $0 \notin Val(T^k)_i$. In other words,

$$val(i_1)_i^k = val(i_1 i_2)_i^k = \cdots = val(i_1 \cdots i_{t-1})_i^k = null.$$

Since $val(i_1 \cdots i_{t-1})_i^k = null$ and $val(i_1 \cdots i_{t-1})_j^k = 0$, process $p_{i_{t-1}}$ crashes during round t and does not send a message to p_i in this round. In the same way, from $val(i_1 \cdots i_l)_i^k = null$ and $val(i_1 \cdots i_l)_{i_{l+1}}^k = 0$, we deduce that p_{i_l} crashes during round $l + 1$ and thus no messages from p_{i_l} arrive at p_i in rounds $l + 1, \ldots, t$. Moreover, p_k has crashed in the first round since it has not sent a message to p_{i_1} in this round ($val(\lambda)_{i_1}^k = 0$). Since $t \geqslant 2$, p_i receives no message from p_k in round t. Therefore, p_i receives no message from $p_k, p_{i_1}, \ldots, p_{i_{t-1}}$ in round t. By definition of T^k, all these processes are distinct. This yields a contradiction with the fact that p_i receives at least $n + 1 - t$ messages in round t. □

Lemma 5.5. *If p_i decides v and p_j decides v' both at round $t + 1$ then $v = v'$.*

Proof. The proof is similar to the proof of Lemma 5.4. □

Theorem 5.6. *For any $t \geqslant 2$, the Tree$_t$ algorithm tolerates t crashes and solves the uniform consensus problem within $t+1$ rounds. Moreover, all correct processes make a decision by the end of round t in all the runs with less than t faulty processes.*

Proof. Termination is obvious, since for any correct process p_i that has not yet decided at the end of round t, *done*$_i$ remains set to *false* up to round $t+1$, and so p_i makes a decision at the end of round $t+1$. Moreover, in a run with less than t failures, any process that is still alive at the end of round t receives messages from at least $n+1-t$ processes at each round, and so makes a decision at round t.

Validity follows from Lemma 5.3.

For uniform agreement, let p_i and p_j be any two processes that decide v and v' at round r and r', respectively. From the algorithm, r and r' are equal to t or $t+1$. There are two cases to consider.

(1) $r = r'$. Then Lemmas 5.4 and 5.5 imply that $v = v'$.

(2) $r \neq r'$. For example, assume $r = t$ and $r' = t+1$. We consider two cases:

(a) Process p_i is still alive when sending messages in round $t+1$. In this case, p_j receives (D, v) from p_i and thus p_j decides $v' = v$.

(b) Process p_i does not send a message to p_j (because it crashes) in round $t+1$. Since p_j does not decide in round t, p_j receives messages from at most $n-t$ processes in this round. Process p_i is among these $n-t$ processes since it is still alive until the end of round t. Therefore, at least $t+1$ processes have crashed in round $t+1$—a contradiction. Thus, case (b) cannot occur.

This proves that p_i and p_j, whether correct or faulty, make the same decision in any possible case. □

5.3. A single algorithm

For each f, $0 \leqslant f \leqslant t$, we have thus exhibited a uniform consensus algorithm that achieves the corresponding lower bound of Theorem 4.1. We strengthen the result by showing that the *EDAUC* and *Tree$_t$* algorithms can be stitched together in a non-mutually destructive way so that the resulting algorithm alone achieves the lower bound for all the values of f.[3]

The *EDAUC* and *Tree$_t$* algorithms are combined in the following way: each process runs *EDAUC* and *Tree$_t$* in parallel. During the $t-1$ first rounds, a process decides according to *EDAUC*, that is if *EDAUC* allows it to decide. Otherwise, at rounds t or $t+1$, a process decides according to *Tree$_t$* unless it receives a (D, v) message from the *EDAUC* algorithm,

[3] Since this paper was first written, the first author and Fabrice Le Fessant have devised a more direct and general "single algorithm" which also solves other agreement problems including non-blocking atomic commitment [5].

in which case it decides v (in other words, an *EDAUC* decision prevails over a *Tree*$_t$ decision). We denote the so-built algorithm by *OptEDAUC*.

Theorem 5.7. *The OptEDAUC algorithm solves the uniform consensus problem, tolerates t crashes, and achieves the lower bounds of Theorem* 4.1.

Proof (*sketched*). Validity and termination are obvious.

For uniform agreement, the only delicate point is to prove that this condition is satisfied even if the decision rules of *EDAUC* and *Tree*$_t$ are both applied in a run. Let p and p' be two processes that decide v and v' at rounds r and r', respectively. Suppose that p decides according to *EDAUC* and p' decides according to *Tree*$_t$; hence, $r' = t$ or $r' = t + 1$. There are two cases to consider.

(1) $r \leqslant t$. According to *EDAUC*, process p sends a (D, v) message to all processes at round r just before deciding. Process p' is alive at the end of round r (indeed $r \leqslant r'$) and receives the (D, v) message from p. Since the decisions according to *EDAUC* prevail, p' decides on v.

(2) $r = t + 1$. In this case, process p has received a (D, v) message from some process q at round t that forces p to decide v at round $t + 1$.

(a) $r' = t$. Since p' has not yet made a decision (according to *EDAUC*) at round t, p' has detected failures at each round $1, \ldots, t - 1$, and so it has detected at least $t - 1$ failures. From the code of *Tree*$_t$, p' observes no new failure at round t; consequently, it receives a message from q, and this message is (D, v). Process p has to decide on v.

(b) $r' = t + 1$. Process p sends a (D, v) message to all processes at round $t + 1$ before making a decision. Upon receiving this message, process p' ought to decide on v.

In each case, we have $v = v'$ as needed. □

6. Discussion

The paper has performed an analysis of time complexities for both consensus and uniform consensus in synchronous systems with crash failures. Our analysis shows that, as for both the byzantine and the omission failure models, uniform consensus is harder than consensus for the crash failure model. It is interesting to note that with crash failures, the difference between these two problems lies in their time complexities, whereas the discrepancy is already noticeable in terms of their solvabilities with byzantine or omission failures.

A result of [7] shows that any algorithm solving a problem specification also solves the uniform version of the specification in most systems that are not synchronous. In particular, this implies that consensus and uniform consensus have the same complexities in such systems. Therefore, our result also shows that when they are achievable, uniformity requirements may force additional costs which depend on the synchrony of the system.

Acknowledgments

We are grateful to Sam Toueg for valuable discussions, and to Idit Keidar and Sergio Rajsbaum for pointing out an inaccuracy in our first proof of Theorem 4.1. It is a pleasure to thank the anonymous referees for their precise comments and suggestions on the first version of this paper, and Michael Merritt and Yoram Moses for the communication of their papers.

References

[1] M. Aguilera, S. Toueg, A simple bivalency-based proof that t-resilient consensus requires $t + 1$ rounds, Inform. Process. Lett. 71 (4) (1999) 155–158.

[2] A. Bar-Noy, D. Dolev, C. Dwork, H.R. Strong, Shifting gears: changing algorithms on the fly to expedite Byzantine agreement, in: Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing, 1987, pp. 42–51.

[3] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison–Wesley, 1987.

[4] T.D. Chandra, S. Toueg, Unreliable failure detectors for asynchronous systems, J. ACM 43 (2) (1996) 225–267.

[5] B. Charron-Bost, F. Le Fessant, Validity conditions in agreement problems and time complexity, Technical Report RR-4526, INRIA-Rocquencourt, August 2002.

[6] B. Charron-Bost, R. Guerraoui, A. Schiper, Synchronous systems and perfect failure detectors: solvability and efficiency issues, in: Proceedings of the International Conference on Dependable Systems and Networks, IEEE, 2000, pp. 523–532.

[7] B. Charron-Bost, S. Toueg, A. Basu, Revisiting safety and liveness in the context of failures, in: Proceedings 11th International Conference on Concurrency Theory, in: Lecture Notes in Comput. Sci., vol. 1877, Springer-Verlag, 2000, pp. 552–565.

[8] R.A. De-Millo, N.A. Lynch, M.J. Merritt, Cryptographic protocols, in: Proceedings of the Fourteenth ACM Symposium on Theory of Computing, ACM Press, 1982, pp. 383–400.

[9] D. Dolev, R. Reischuk, H.R. Strong, Early stopping in Byzantine agreement, J. ACM 37 (4) (1990) 720–741.

[10] D. Dolev, H.R. Strong, Polynomial algorithms for multiple processor agreement, in: Proceedings of the Fourteenth ACM Symposium on Theory of Computing, ACM Press, 1982, pp. 401–407.

[11] C. Dwork, N.A. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323.

[12] C. Dwork, Y. Moses, Knowledge and common knowledge in a Byzantine environment: crash failures, Inform. and Comput. 88 (2) (1990) 156–186.

[13] M.J. Fischer, N.A. Lynch, A lower bound for the time to assure interactive consistency, Inform. Process. Lett. 14 (1982) 183–186.

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

[15] E. Gafni, Rounds-by-rounds fault detectors: unifying synchrony and asynchrony, in: Proceedings of the Seventeenth ACM Symposium on Principles of Distributed Computing, 1998, pp. 143–152.

[16] A. Gopal, Fault-tolerant broadcasts and multicasts: the problem of inconsistency and contamination, PhD thesis, Cornell University, January 1992.

[17] R. Guerraoui, Revisiting the relationship between non-blocking atomic commitment and consensus, in: 9th International Workshop on Distributed Algorithms, in: Lecture Notes in Comput. Sci., vol. 972, Springer-Verlag, 1995, pp. 87–100.

[18] V. Hadzilacos, On the relationship between the atomic commitment and consensus problems, in: Fault Tolerant Distributed Computing, in: Lecture Notes in Comput. Sci., vol. 448, Springer-Verlag, 1987, pp. 201–208.

[19] V. Hadzilacos, S. Toueg, A modular approach to fault-tolerant broadcasts and related problems, Technical Report TR 94-1425, Cornell University, Department of Computer Science, May 1994.

[20] I. Keidar, S. Rajsbaum, A simple proof of the uniform consensus synchronous lower bound, Inform. Process. Lett. 85 (1) (2003) 47–52.

[21] L. Lamport, The weak byzantine generals problem, J. ACM 30 (3) (1983) 668–676.

[22] L. Lamport, Lower bounds on consensus, Unpublished manuscript, March 2000.

[23] L. Lamport, Lower bounds for asynchronous consensus, in: Proceedings of the International Workshop on Future Directions in Distributed Computing, in: Lecture Notes in Comput. Sci., vol. 2584, Springer-Verlag, 2002, pp. 22–23.

[24] L. Lamport, M. Fischer, Byzantine generals and transaction commit protocols, Technical Report 62, SRI International, April 1982.

[25] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Trans. Programm. Languages Systems 4 (3) (1982) 382–401.

[26] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann, 1996.

[27] M.J. Merritt, Unpublished notes, 1985.

[28] Y. Moses, S. Rajsbaum, A layered analysis of consensus, SIAM J. Comput. 31 (4) (2002) 989–1021.

[29] G. Neiger, S. Toueg, Automatically increasing the fault-tolerance of distributed algorithms, J. Algorithms 11 (3) (1990) 374–419.

[30] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, J. ACM 27 (2) (1980) 228–234.

[31] K.J. Perry, S. Toueg, Distributed agreement in the presence of processor and communication faults, IEEE Trans. Software Engrg. 12 (3) (1986) 477–482.

[32] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak, C.B. Weinstock, SIFT: design and analysis of a fault-tolerant computer for aircraft control, Proc. IEEE 66 (10) (1978) 1240–1255.