Revisiting Simultaneous Consensus with Crash Failures

Y. Moses, Michel Raynal

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION INTERNE

N^{o} 1885

REVISITING SIMULTANEOUS CONSENSUS WITH CRASH FAILURES

Y. MOSES M. RAYNAL

ISSN 1166-8687

IRISA

CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

Revisiting Simultaneous Consensus with Crash Failures

Y. Moses[*] M. Raynal[**]

Systèmes communicants
Projet ASAP

Publication interne n ˚ 1885 — Mars 2008 — 17 pages

Abstract: This paper addresses the “consensus with simultaneous decision” problem in a synchronous system prone to t process crashes. This problem requires that all the processes that do not crash decide on the same value (consensus) and that all decisions are made during the very same round (simultaneity). So, there is a double agreement, one on the decided value (data agreement) and one on the decision round (time agreement). This problem was first defined by Dwork and Moses who analyzed it and solved it using an analysis of the evolution of states of knowledge in a system with crash failures. The current paper presents a simple algorithm that optimally solves simultaneous consensus. Optimality means in this case that the simultaneous decision is taken in each and every run as soon as any protocol decides, given the same failure pattern and initial value. The design principle of this algorithm is simplicity, a first-class criterion. A new optimality proof is given that is stated in purely combinatorial terms.

Key-words: Consensus, Distributed algorithm, Fault-tolerance, Round-based computation, Simultaneous decision, Synchronous message-passing system.

(Résumé : tsvp)

[*] Department of Electrical Engineering, Technion, Haifa, 32000 Israel moses@ee.technion.ac.il
[**] IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France raynal@irisa.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Décision simultanée en environement synchrone avec crash de processus

Résumé : Ce rapport présente un algorithme de consensus pour un système synchrone avec crash de processus, dans lequel les processus qui décident le font à la même ronde de calcul. L'accent erst mis sur la simplicité de conception de cet algorirthme.

Mots clés : Système synchrone, algorithme distribué, consensus, crash de processus, décision simultaée, modèle de calcul fondé sur les rondes, système synchrone.

1 Introduction

The consensus problem Fault-tolerant systems often require a means by which processes or processors can arrive at an exact mutual agreement of some kind [15]. If the processes defining a computation have never to agree, that computation is actually made up of a set of independent computations, and consequently is not an inherently distributed computation. The agreement requirement is captured by the *consensus* problem that is one of the most important problems of fault-tolerant distributed computing. It actually occurs every time entities (usually called agents, processes -the word we use in the following-, nodes, sensors, etc.) have to agree. The consensus problem is surprisingly simple to state: each process is assumed to propose a value, and all the processes that are not faulty have to agree/decide (termination), on the same value (agreement), that has to be one of the proposed values (validity). The failure model considered in this paper is the process crash model.

While consensus is impossible to solve in pure asynchronous systems despite even a single process crash [6] ("pure asynchronous systems" means systems in which there is no upper bound on process speed and message transfer delay), it can be solved in synchronous systems (i.e., systems where there are such upper bounds) whatever the number n of processes and the number t of process crashes ($t < n$).

An important measure for a consensus algorithm is the time it takes for the non-faulty processes to decide. As a computation in a synchronous system can be abstracted as a *sequence of rounds*, the time complexity of a synchronous consensus algorithm is measured as the minimal number of rounds (R_t) a process has to execute before deciding, in the worst case scenario. It has been shown (see, e.g., in [5, 12]) that $R_t = t + 1$. Moreover, that bound is tight: there exist algorithms (e.g., see [1, 9, 16]) where no process ever executes more than R_t rounds (these algorithms are thus optimal with respect to that bound).

While $t + 1$ rounds are needed in the worst case scenario, the major part of the executions have few failures or are even failure-free. So, an important issue is to be able to design *early deciding* algorithms, i.e., algorithms that direct the processes to decide "as early as possible" in good scenarios. Let f, $0 \leq f \leq t$, be the number of actual process crashes in an execution. It has been shown that the lower bound on the number of rounds is then $R_{t,f} = \min(f + 2, t + 1)$ (e.g., [2, 12, 17]). As before, this bound is tight: algorithms in which no process ever executes more than $R_{t,f}$ exist (e.g., see [2, 7, 16]).

Simultaneous decision Consensus agreement is a data agreement property, namely the processes have to agree on the same value. According to the actual failure pattern, and the way this pattern is perceived by the processes, it is possible for several processes to decide at distinct rounds. The only guarantee lies in the fact that this round can be bounded by R_t (or $R_{t,f}$).

This uncertainty on the set of round numbers at which the processes decide, can be a serious drawback for the real-time oriented applications where agreement is required, not only on the decided value, but also on the time the decision is taken. More precisely, these applications require that the processes decide on the same value (*data agreement*), during the very same round (*time agreement*). This property is also called *simultaneous decision*.

Among the algorithms that ensure simultaneous decision, there are trivially all the "classical" consensus algorithms where all the processes that do not crash decide systematically at the end of the round $R_t = t + 1$. This observation suggests immediately the following question: "As far as the simultaneous decision property is concerned, are there early deciding algorithms, i.e., algorithms whose maximal number of rounds in the worst case scenario can be determined from f (and t)?" Unfortunately, it is shown in [2] that the answer to that question is negative: $R_t = t + 1$ rounds is the best that can be done when both the parameters t and f are considered. At first glance, this can appear as counter-intuitive as it states that $t + 1$ is a bound for simultaneous decision whatever the value of f (i.e., even when no process crashes)!

Early simultaneous decision So, given an execution, a more refined analysis requires to consider not the parameters t and f, but t and the failure pattern that actually occurs in the execution. The previous question can now be translated as follows: "Which are the failure patterns that force a simultaneous decision consensus algorithm to decide in k rounds for $2 \leq k \leq t + 1$". This question was posed and answered in [3] where a bound is stated and proved (this bound is denoted $RS_{t,F}$ in the following; F stands for the failure pattern that occurs in the considered run).

To better understand the intuition that underlies the bound $RS_{t,F}$, let us consider the particular failure pattern where t processes have crashed before the execution starts. The $n - t$ remaining processes define consequently a failure-free system. During the first round each non-crashed process can learn that, from then on, it is in a failure-free system of $n - t$ processes, and consequently all the processes can exchange their view of the system and decide the same value at the end of the second round. More generally, what makes things easier is when many crashes occur early in the computation. Roughly speaking, this is because a crash is stable, while the property "a process has not crashed" is not a stable property. This instability property and the occurrence of only a few crashes, makes an agreement on an early round for a simultaneous decision more difficult to obtain.

The previous discussion suggests that determining the smallest round at which the processes can simultaneously decide should take into account the pairs (round number, number of processes perceived as crashed until that round). This intuition is formalized in [3] as follows. Let $C[r]$ be the set of the processes that are seen as crashed by (at least) one of the processes that survive (i.e., do not crash before the end of) round r. For any r, let $d_r = \max(0, |C[r]| - r)$; as we shall see, d_r represents the number of rounds that could be saved with respect to the worst case $t + 1$ bound, thanks to the failures that occurred and were seen by at least one process that terminate round r. Let $D = \max_{r \geq 0}(d_r)$; D represents the best saving in terms of rounds. Notice that the values of d_r and D in a given run are determined only by its failure pattern F. It is shown in [3] that the smallest round number at the end of which a common decision can be simultaneously taken is $RS_{t,F} = (t + 1) - D$ (where $D = D(F)$). The quantity D is considered in [3] to be the *waste* inherent in the failure pattern F, since it specifies the number of rounds the adversary has "lost" in its quest to delay decision for as long as possible. This optimality proof is established in [3] from the theory of distributed (implicit) knowledge and common knowledge (and the way distributed knowledge becomes common knowledge in a distributed system) [4, 8]. An optimal binary consensus algorithm that directs the processes to decide simultaneously at the end of the round $RS_{t,F} = (t + 1) - D$ is also described in [3]. Its construction is derived based on a the same knowledge-based analysis.

Content of the paper This paper presents a simple construction of an optimal simultaneous multi-valued consensus algorithm, that can be seen as a variant that revisits the algorithm presented in [3]. This variant is based on concepts and proofs introduced in [10, 11]. Here, the aim is not to design a brand new algorithm, but to construct a simple algorithm from a few definitions and simple observations. Moreover, a simpler matching lower bound proving the optimality of this algorithm is presented. The main criterion is design simplicity. Interestingly, not only the design, but also the proofs of the algorithm are simple. While the connection to knowledge is hinted at here and there, the development and proofs are purely combinatorial. We hope that the relative simplicity of the development in this paper will make the simultaneous agreement property and the simple (and elegant) concepts its implementation relies on more broadly accessible.

The paper is made up of 6 sections. The computation model and the problem are introduced in Section 2. The algorithm is presented in Section 3, while its proof is given in Section 4. Section 5 presents a new lower-bound proof (simplifying that of [3]) of the optimality of the algorithm presented in Section 3. Finally, Section 6 states a few concluding remarks.

2 Model and problem specification

2.1 Computation model

Round-based synchronous system The system model consists of a finite set of processes, namely, $\Pi = \{p_1, \dots, p_n\}$, that communicate and synchronize by sending and receiving messages through channels. (Sometimes we also use p and q to denote processes.) Every pair of processes is connected by a bi-directional reliable channel (which means that there is no creation, alteration, loss or duplication of message).

The system is *round-based synchronous*. This means that each execution consists of a sequence of *rounds*. Those are identified by the successive integers $1, 2,$ etc. For the processes, the current round number appears as a global variable r that they can read, and whose progress is managed by the underlying system. A round is made up of three consecutive phases:

- A send phase in which each process sends the same message to all the processes (including itself).
- A receive phase in which each process receives messages.
  The fundamental property of the synchronous model lies in the fact that a message sent by a process p_i to a process p_j at round r, is received by p_j at the very same round r.

- A computation phase during which each process processes the messages it received during that round and executes local computation.

Process failure model A process is *faulty* during an execution if its behavior deviates from that prescribed by its algorithm, otherwise it is *correct*. We consider here the crash failure model, namely, a faulty process stops its execution prematurely. After it has crashed, a process does nothing. If a process crashes in the middle of a sending phase, only a subset of the messages it was supposed to send might actually be received.

As already indicated, the model parameter t ($1 \leq t < n$) denotes an upper bound on the number of processes that can crash in a run. A *failure pattern* F is a list of at most t triples $\langle q, k_q, B_q \rangle$. A triple $\langle q, k_q, B_q \rangle$ states that the process q crashes while executing the round k_q (hence, it sends no messages after round k_q), while the set B_q denotes the set of processes that do not receive the message sent by q during the round k_q.

2.2 The simultaneous consensus problem

The problem has been informally stated in the introduction: every process p_i *proposes* a value v_i (called its *initial value*) and the processes have to *decide*, during the very same round, on the same value that has to be one of the proposed values. This can be stated as a set of four properties that any algorithm solving the problem has to satisfy.

- Decision. Every correct process decides.
- Validity. A decided value is a proposed value.
- Data agreement. No two processes decide different values.
- Simultaneous decision (or Time agreement). No two processes decide at distinct rounds.

Given a set V of two or more values, an *input configuration* is an assignment $I : \Pi \rightarrow V$ of an initial value v_i to each process p_i. We are considering the simultaneous consensus problem under the assumption that all $|V|^n$ input configurations are possible.

Traditional consensus algorithms for the crash failure model are guaranteed to decide within at most $t+1$ rounds [2, 15]. Any such algorithm can be converted into a simultaneous consensus algorithm by delaying any early decision and having all deciding processes decide only in round $t+1$. Dwork and Moses showed that simultaneous decision can often be obtained much earlier than that [3]. Although every algorithm will have runs that decide in $t+1$ rounds, simultaneous decision can be obtained as early as the second round in some cases. The goal, then, is to design a *simple* algorithm that direct the processes to decide both as early as possible and simultaneously.

3 A simple optimal algorithm

3.1 Preliminary definitions

As the system model requires each process to send a message to all the processes at each round, process failures can be easily detected, and this detection is done as soon as possible. In addition to this very simple failure detection mechanism, the algorithm is based on other simple notions, namely, the notions of *clean round* (introduced in [3]), and the notion of *horizon* (introduced in [10]).

Failure discovery The failure of a process q is *discovered* (for the first time) in round r if r is the first round such that there is a process p that (1) does not receive a round r message from q, and (2) survives (i.e., completes without crashing) round r.

Clean round A round r is *clean* if no process is discovered faulty for the first time in that round. This means that a process that crashes during a clean round r has sent its round r message to all the processes that proceed to the round $r+1$.

Let us call an algorithm *symmetric* if a process never sends different messages to distinct processes in the same round. The following property is an immediate consequence of the previous definitions. [1]

[1] In a precise sense, a clean round can be used to ensure that the knowledge of the various processes is identical. Once this happens the processes are in agreement about initial values. They then need to discover this and coordinate their decisions.

Property 1 *In a symmetric algorithm, if round r is clean, then all the processes that proceed to the round $r+1$ received, during the round r, messages from the same set of processes (including at least all of them).*

Let us observe that a clean round is not necessarily a failure-free round. It is possible that a process p crashes in a clean round r but no process active at the end of r has noticed its crash (p has crashed after its sending phase and before the end of the round r, or more generally p as crashed during r after sending its round r message at least to the processes that survive round r). Similarly a failure-free round is not necessarily clean. As an example a failure-free round $r+1$ that follows a clean round r during which a crash occurred is not clean.

Horizon [10] Given a process p_i and a round $r \geq 1$, let x be the greatest number of process crashes that occurred between the round 1 and the round $r-1$ (included) and are known by p_i (to have crashed in the first $r-1$ rounds) by the end of r.

The value $h_i(r) = r + t - x$ is called the *horizon* of p_i at round r. We have $h_i(1) = t + 1$. If three crashes occurred by the end of the first round and are reported to p_i during the second round, we have $h_i(2) = t - 1$.

As we will see, the horizon notion (of a process p_i at round r) is a key notion to determine the smallest round at the end of which the same value can be simultaneously decided. The following simple theorem (that will be exploited in the presentation of the algorithm) explains why this notion is crucial.

Theorem 1 *Let x be defined as indicated above, and p_i a process that survives round r. There is a clean round y such that $r \leq y \leq h_i(r) = r + t - x$.*

Proof Let us first observe that, as at least x processes have been discovered as faulty between the round 0 and the round $r-1$ (included), at most $t-x$ processes can be discovered as faulty between the round r (included) and the round $r+t-x$ (included). But there are $t-x+1$ rounds from r to $r+t-x$, from which we conclude that at least one of these rounds is clean. $\Box_{Theorem\ 1}$

3.2 Description of the algorithm

Local variables Each process p_i manages the following local variables. Some variables are presented as belonging to an array. This is only for notational convenience, as such array variables can be implemented as simple variables.

- est_i contains, at the end of r, p_i's current estimate of the decision value. Its initial value is v_i, the value proposed by p_i.
- $f_i[r]$ denotes the set of processes from which p_i has not received a message during the round r. (So, this variable is the best current estimate that p_i can have of the processes that have crashed.)
  Let $\overline{f_i[r]} = \Pi \setminus f_i[r]$ (i.e., the set of processes from which p_i has received a round r message).
- $f'_i[r-1]$ is a value computed by p_i during the round r, but that refers to crashes that occurred up to the round $r-1$ (included), hence the notation. It is the value $\bigcup_{p_j \in \overline{f_i[r]}} f_j[r-1]$, which means that $f'_i[r-1]$ is the set of processes that were known as crashed at the end of the round $r-1$ by at least one of the processes from which p_i has received a round r message. This value is computed by p_i during the round r. As a process p_i receives its own messages, we have $f_i[r-1] \subseteq f'_i[r-1]$.
- $bh_i[r]$ represents the best (smallest) horizon value known by p_i at round r. It is p_i's best estimate of the smallest round for a simultaneous decision. Initially, $bh_i[0] = h_i(0) = t+1$.

Process behavior Each process p_i not crashed at the beginning of r sends to all the processes a message containing its current estimate of the decision value (est_i), and the set $f_i[r-1]$ of processes it currently knows as faulty. After it has received the round r messages, p_i computes the new value of est_i and the value of $bh_i[r]$. The new value of est_i is the smallest of the estimates values it has seen so far. As far as the value of $bh_i[r]$ is concerned, we have the following.

- The computation of $bh_i[r]$ has to take into account $h_i(r)$. This is required to benefit from Theorem 1 that states that there is a clean round y such that $r \leq y \leq h_i(r)$. When this clean round will be executed, any two processes p_i and p_j that execute it will have $est_i = est_j$, and (as they will receive messages from the same set of processes, see Property 1) will be such that $f'_i[r-1] = f'_j[r-1]$. It follows that, we will have $h_i(y) = h_j(y)$, thereby creating correct "seeds" for determining the smallest round for a simultaneous decision. This allows the processes to determine rounds at which they can simultaneously decide.

- As we are looking for the first round where a simultaneous decision is possible, $bh_i[r]$ has to be set to $\min\big(h_i(0), h_i(1), \dots, h_i(r)\big)$ i.e., $bh_i[r] = \min\big(bh_i[r-1], h_i(r)\big)$.

Finally, according to the previous discussion, the algorithm directs a process p_i to decide at the end of the first round r that is equal to the best horizon currently known by p_i, i.e., when $r = bh_i[r]$.

The resulting algorithm is presented in Figure 1, where $h_i(r)$ (see line 08) is expressed as a function of $r-1$ to emphasize the fact that it could be computed at the end of the round $r-1$ by an external omniscient observer. The local boolean variable decided is used only to prove the optimality of the algorithm (see Section 5). Its suppression does not alter the algorithm.

```
algorithm PROPOSE($v_i$):
(01)  $est_i \leftarrow v_i$; $bh_i[0] \leftarrow t+1$; $f_i[0] \leftarrow \emptyset$; decided $\leftarrow false$;      % initialization %
(02)  when $r = 1, 2, \dots$ do      % $r$: round number %
(03)  begin round
(04)      send $(est_i, f_i[r-1])$ to all;      % including $p_i$ itself %
(05)      let $f'_i[r-1]$ = the union of the $f_j[r-1]$ sets received during $r$;
(06)      let $f_i[r]$ = the set of processes from which $p_i$ has not received a message during $r$;
(07)      $est_i \leftarrow \min($ all the $est_j$ received during $r)$;
(08)      let $h_i(r) = (r-1) + (t+1-|f'_i[r-1]|)$;
(09)      $bh_i[r] \leftarrow \min\big(bh_i[r-1], h_i(r)\big)$;
(10)      if $r = bh_i[r]$ then decided $\leftarrow true$; return $(est_i)$ end if
(11)  end round
```

Figure 1: Optimal simultaneous decision despite up to t crash failures (code for p_i)

4 Proof of the algorithm

Lemma 1 Validity property. *A decided value is a proposed value.*

Proof The proof is an immediate consequence of the initialization of the est_i local variables (line 01), the reliability of the channels, and the $\min()$ operation used at line 07. $\Box_{Lemma\ 1}$

Lemma 2 *Let p_i be a correct process. $\forall r \geq 0$ we have $h_i(r) \geq r$.*

Proof Since the processes in the set $f'_i[r-1]$ are processes that have crashed by the end of the round $r-1$, it follows that $t - |f'_i[r-1]| \geq 0$. Consequently, $h_i(r) = r + t - |f'_i[r-1]| \geq r$. $\Box_{Lemma\ 2}$

Notation: Considering an arbitrary execution, let p_i be a process that is correct in that execution.

- Let $BH_i = \min_{r \geq 0} h_i(r)$. BH_i is the smallest value ever attained by the function $h_i(r)$, i.e., the smallest horizon value determined by p_i.
- Let $L_i = \max(\{r \mid h_i(r) = BH_i\})$. L_i is the last round whose horizon value is BH_i.

It follows from these definitions that if $L' > L_i$ then $h_i(L') > h_i(L_i)$.

Lemma 3 *Let $t < n$. The round L_i is a clean round (i.e., no process is discovered faulty for the first time in that round).*

Proof Assume, by way of contradiction, that L_i is not clean (recall that p_i is a correct process). This means there is a process p_z that is seen faulty for the first time in round L_i by some process p_y. Notice that $p_z \notin f'_i[L_i - 1]$ since p_z was not discovered faulty in the previous rounds. There are two cases.

- Case 1: p_i receives a message from p_y in round $L_i + 1$.
  (This case includes the case where p_i and p_y are the same process). As p_y does not receive a message from p_z during L_i, and a crash is stable, we have $p_z \in f_y[L_i]$. Moreover, due to the case assumption, and the fact that the round $L_i + 1$ message from p_y to p_i carries $f_y[L_i]$, it follows that $f'_i[L_i]$ contains $f'_i[L_i - 1] \cup \{p_z\}$. Consequently, $|f'_i[L_i]| > |f'_i[L_i - 1]|$. It follows that $h_i(L_i + 1) \leq h_i(L_i)$, contradicting the definition of L_i.

- Case 2: p_i does not receive a message from p_y in round $L_i + 1$.
  In that case, both p_z and p_y are seen faulty for the first time by p_i during the round $L_i + 1$. So, $f_i[L_i + 1]$ contains $f'_i[L_i - 1] \cup \{p_y, p_z\}$. Since $f'_i[L_i + 1]$ (computed by p_i during the round $L_i + 2$) contains $f_i[L_i + 1]$, we have $|f'_i[L_i + 1]| \geq |f'_i[L_i - 1]| + 2$. Thus, we have

$$\begin{array}{rcl} h_i(L_i + 2) & = & (L_i + 2) + t - \ |f'_i[L_i + 1]|, \\ & \leq & (L_i + 2) + t - (|f'_i[L_i - 1]| + 2), \\ & = & L_i + t - \ |f'_i[L_i - 1]|, \\ & = & h_i(L_i), \end{array}$$

  which again contradicts the definition of L_i.

$\square_{Lemma\ 3}$

Lemma 4 *Let $t < n$. Every correct process decides. Moreover, all processes that decide do so in the same round and decide on the same value.*

Proof
Decision property. Let us consider a correct process p_i. Notice that, due to the initialization and line 09 we have $\forall r\ :\ bh_i[r] \leq t + 1$, from which we conclude $BH_i \leq t + 1$. So, to prove that p_i decides we have to show that p_i does not miss the test $r = BH_i$ at line 10. This could happen if the first round ℓ such that $bh_i[\ell - 1] > BH_i$ and $bh_i[\ell] = BH_i$ is such that $\ell > BH_i$. We prove that this cannot happen.

Let us observe that, due to Lemma 2, we have $h_i(\ell) \geq \ell$. It then follows from $bh_i[\ell - 1] > BH_i$, $h_i(\ell) \geq \ell$, $bh_i[\ell] = BH_i$, and line 09, that $BH_i = bh_i[\ell] = \min(bh_i[\ell - 1], h_i(\ell)) = h_i(\ell) \geq \ell$, i.e., $BH_i \geq \ell$, which establishes the result. It follows that p_i decides no later than round $t + 1$.

Simultaneous decision for the correct processes. We first show that no two correct processes p_i and p_j decide at distinct rounds. Due to the algorithm, if p_i and p_j decide, they decide at round BH_i and BH_j, respectively. We show that $BH_i = BH_j$. Due to Lemma 3, the round L_i is clean. Hence, during the round L_i, p_j receives the same messages that p_i receives (Property 1). Thus $f'_i[L_i - 1] = f'_j[L_i - 1]$ and consequently, $h_i(L_i) = h_j(L_i)$. Since $bh_j[L_i] \leq h_j(L_i)$ by line 09, it follows that $bh_j[L_i] \leq bh_i[L_i] = BH_i$, and thus $BH_j \leq BH_i$. By symmetry the same reasoning yields $BH_i \leq BH_j$, from which it follows that $BH_i = BH_j$. This proves that no two correct processes decide at distinct rounds.[2]

Simultaneous decision for the faulty processes. BH being the round at which the correct processes decide, let us now consider the case of a faulty process p_j. As p_j behaves as a correct process until it crashes, and as the correct processes decide in the same round BH, it follows that no faulty process decides before BH, and if p_j executes line 10 of round BH, it does decide as if it was a correct process.

Data agreement property**.** The fact that no two processes decide different values comes from the existence of the clean round L_i that appears before a process decision. During that round, all the processes that are alive at the end of this round have received the same set of estimate values (Property 1), and selected the smallest of them. It follows that, from the end of that round, there is a single estimate value in the system, which proves the data agreement property.

$\square_{Lemma\ 4}$

Definitions This paragraph recalls notions and results that have already been stated in the introduction. Given an execution, let F be the failure pattern that occurs in that execution.

- Let $S[r]$ be the set of processes that survive (i.e., complete) round r.
- Let $C[r] = \bigcup_{p_i \in S[r]} f_i[r]$, i.e., the set of the processes that are known to have crashed by at least one of the processes that survives round r. Observe that $f'_i[r] \subseteq C[r]$, for any $p_i \in S[r]$.

[2] In [3, 8] it is shown that before performing such a simultaneous action the processes must attain common knowledge that they are doing so. In particular, they must have common knowledge that the decided value v is one of the initial values in the run.

- Let $D = \max_{r \geq 0}(d_r)$ where $d_r = |C[r]| - r$. (In the introduction, D has been called the *waste* inherent in F, i.e., the number of rounds the adversary has lost in his quest to delay decision for as long as possible.)

Notice that $D \geq 0$, since $C[0] = 0$ and $D \geq d_0 = C[0] - 0 = 0$. The following optimality results are shown in [3]. The smallest number of rounds $RS_{t,F}$ that any simultaneous decision consensus algorithm can achieve is $RS_{t,F} = (t+1) - D$ when $t < n-1$, and $RS_{t,F} = t - D$ when $t = n-1$.

Theorem 2 *Let $t < n$. The algorithm described in Figure 1 solves the consensus problem with simultaneous decision. In a run with failure pattern F, decision is reached in round $t+1-D$ where $D = D(F)$ is the waste inherent in F.*

Proof The proof of the validity, decision, simultaneous decision and data agreement properties follow from the Lemmas 1 and 4. We now show that the decision is obtained in round $t+1-D$. Let us consider an arbitrary run of the algorithm. It follows from Lemma 4 that $BH_i = BH_j$ for any pair of processes p_i and p_j that decide. Let BH denote this round. The proof of the claim amounts to showing that $BH \leq t+1-D$ and $BH \geq t+1-D$.

Let p_i be a process that decides and R the last round such that $|C[R]| - R = D$ (i.e., $|C[R+x]| - (R+x) < D = |C[R]| - R$, for any $x > 0$). Let us observe that, due to the lines 08-10 of the algorithm, BH is attained at the round numbers that make the function $h_i(r) = (r-1) + t - |f'_i[r]| + 1$ minimal. Moreover, it follows from the definition of D and R that $|C[R+1]| \leq |C[R]|$. Since $C[R] \subseteq C[R+1]$, it follows that $C[R] = C[R+1]$, i.e., no new process failure is discovered in round $R+1$, so the round $R+1$ is clean and we have $|f'_i[R]| = |C[R]|$. Due to line 08 of the round $R+1$, we have $h_i(R+1) = R+t+1-|f'_i[R]| = (t+1) - (|f'_i[R]| - R) = t+1-D$, from which we conclude $BH \leq t+1-D$.

For the other direction, let us recall that, due to Lemma 3, the round $L_i > 0$ is clean. It follows that $f'_i[L_i - 1] = C[L_i - 1]$, since any p_i hears in round L_i from all processes that survived round $L_i - 1$. Therefore, $BH = t+1-(|f'_i[L_i-1]| - (L_i - 1)) = t+1-(|C[L_i-1]| - (L_i-1)) = t+1-d_{(L_i-1)} \geq t+1-D$, which completes the proof of the theorem. $\Box_{Theorem\ 2}$

5 On the optimality of the algorithm: $t+1-D$ is a lower bound

This section proves that the PROPOSE algorithm is optimal: In a synchronous system prone to up to t process crashes (with $t < n-1$), there is no deterministic algorithm that can ever solve the simultaneous consensus problem in fewer than $t+1-D$ rounds. The proof given here is new (and simpler than the first proof given in [3]). It relies on notions introduced in [11]. It also uses notations introduced in Section 4.

The problem of simultaneous consensus is closely related to the knowledge-theoretic notion of similarity among runs at a given time. This notion is captured by the following definitions. For later use, these definitions are made with respect to an arbitrary round-based synchronous deterministic algorithm P (they consequently apply in particular to the PROPOSE algorithm described in Figure 1).

5.1 Preliminary definitions and lemmas

For ease of exposition, the runs of an arbitrary deterministic algorithm P are denoted by σ, σ', etc. $S[r,\sigma]$ denotes the set of processes that survive round r of σ, while $ls(p,r,\sigma)$ denotes the local state of p at the end of r in the run σ (i.e., its set of local variables and their current values).

Definition 1 *Given a deterministic algorithm P, a process p, and a round r,* the runs σ and σ' of P are indistinguishable to p after round r *(denoted $\sigma \overset{r}{\sim}_p \sigma'$) if both (i) $p \in S[r,\sigma] \cap S[r,\sigma']$ (i.e., p has survived round r in both runs), and (ii) $ls(p,r,\sigma) = ls(p,r,\sigma')$.*

Definition 2 *The runs σ and σ' are* connected *at the end of round r, denoted $\sigma \overset{r}{\approx} \sigma'$, if there is a sequence of runs and processes such that $\sigma = \sigma_0 \overset{r}{\sim}_{p_0} \sigma_1 \overset{r}{\sim}_{p_1} \cdots \overset{r}{\sim}_{p_{k-1}} \sigma_k = \sigma'$.*

In other words, an undirected graph $\mathsf{G}(P,r)$ (in short $\mathsf{G}(r)$) can be associated with each round r of a protocol P. This graph is called P's *similarity graph* for round r. Its vertices are the runs of P and there is an edge connecting σ and σ' if there is a process q such that $\sigma \overset{r}{\sim}_q \sigma'$. It is easy to see that $\sigma \overset{r}{\approx} \sigma'$ holds if σ and σ' belong to the same connected component in $\mathsf{G}(r)$. Because $\mathsf{G}(r)$ is undirected, being connected ($\overset{r}{\approx}$) is an equivalence relation.[3] Moreover, observe that if we can show that some property A is maintained under $\overset{r}{\sim}_q$ for all $q \in \Pi$, then whenever σ has property A and $\sigma \overset{r}{\approx} \sigma'$, we are guaranteed that σ' has property A as well.

Lemma 5 *Let σ and σ' be runs of a deterministic algorithm P that solves simultaneous consensus. If some process decides on value v in round r of σ and $\sigma \overset{r}{\approx} \sigma$, then the processes in $S[r, \sigma']$ decide the same value v in the same round r of σ'.*

Proof It suffices to show the claim for any two runs σ, σ' such that $\sigma \overset{r}{\sim}_q \sigma'$ for some $q \in S[r, \sigma] \cap S[r, \sigma']$. In this case, q decides v in round r of σ. Because P is deterministic and q has the same local state at the end of round r of σ and σ', q decides v at the end of round r in σ'. Finally, as P solves simultaneous consensus (lemma assumption) it follows that all processes in $S[r, \sigma']$ decide v in round r (and no other process decides a different value in a different round). $\Box_{Lemma\ 5}$

An immediate consequence of Lemma 5 is captured by the following corollary.

Corollary 1 *Let P be a deterministic algorithm that solves simultaneous consensus. If σ' is a run of P such that (1) no initial value in σ' is v, and (2) $\sigma \overset{r}{\approx} \sigma'$, then no process can decide v in round r of σ.*

Proof Since $n > t$, the set $S[r, \sigma']$ is nonempty. By Lemma 5, if some process q decides v in round r of σ, the processes in $S[r, \sigma']$ decide v in round r of σ'. But this contradicts the Validity property of the simultaneous consensus algorithm P, since the decision value v is not one of the initial values in σ'. $\Box_{Corollary\ 1}$

5.2 A full-information algorithm

For the purpose of proving optimality, we make use of a *full-information* algorithm, denoted FIP and described in Figure 2.

The algorithm In the first round, each process sends its initial value v_i to all processes (including to itself). The algorithm then constructs an array *inp*$_i[1..n]$ containing the incoming message from each of the processes (itself included). If p_i does not receive a message from p_j then it sets *inp*$_i[j]$ to the default value $\perp$. In each of the later rounds, every process p_i first sends *inp*$_i$ to all others, and then uses the incoming messages of the current round to construct an updated array *inp*$_i$ in the same way as in the first round. The local state of the process at the end of round r is identified simply with the contents of its array *inp*$_i$.

This algorithm is introduced in order to establish, for each failure pattern F, times at which simultaneous consensus cannot be reached by any algorithm whatsoever. Optimality will then be established by showing that the PROPOSE algorithm described in Figure 1 decides as soon as possible, for each and every possible failure pattern (and initial configuration).

```
algorithm FIP:
for $j \in \{1, \dots, n\} \setminus \{i\}$ do $inp_i[j] \leftarrow \perp$ end for; $inp_i[i] \leftarrow v_i$;
when $r = 1, 2, \dots$ do
begin round
    send $inp_i$ to all;        % including to $p_i$ itself %
    for $j \in \{1, \dots, n\}$ do
        $inp_i[j] \leftarrow$ message received from $p_j$ during $r$ if any, otherwise $\perp$
    end for
end round
```

Figure 2: The full-information algorithm FIP (code for p_i)

Observe that a deterministic algorithm P, an initial configuration I (set of initial values), and a failure pattern F determine a run $\sigma = P(I, F)$ of P.

[3] In the knowledge terminology, process q *knows* a fact A at the end of round r in σ if it is true of all runs σ' satisfying $\sigma' \overset{r}{\sim}_q \sigma$; it is *common knowledge* there if A holds at all runs $\sigma' \overset{r}{\approx} \sigma$. Thus, the set of runs connected to σ determines what is common knowledge in σ at the end of round r.

Definition 3 *A run σ of P* corresponds to *a run ρ of an algorithm P' if, for some initial configuration I and failure pattern F, it is the case that $\sigma = P(I,F)$ and $\rho = P'(I,F)$.*

The next lemma shows, in a precise sense, that the connected components of the similarity graph for Fip refine those of any other deterministic algorithm.[4]

Lemma 6 *Let P be a deterministic algorithm for simultaneous consensus. Let us assume that the runs σ and σ' of P correspond to the runs ρ and ρ' of* Fip*, respectively. Then (i) if $\rho \overset{r}{\sim}_q \rho'$ then $\sigma \overset{r}{\sim}_q \sigma'$, and (ii) if $\rho \overset{r}{\approx} \rho'$ then $\sigma \overset{r}{\approx} \sigma'$.*

Proof Let us consider the runs σ and σ' of P and the corresponding runs ρ and ρ' of Fip. Since $\overset{r}{\approx}$ is the transitive closure of the relations $\{\overset{r}{\sim}_p\}_{p\in\Pi}$, claim (ii) follows from (i). Consequently, it suffices to prove claim (i). Let us observe that, due to the definition of “corresponds to”, the fact that σ corresponds to ρ implies that $S[r,\sigma] = S[r,\rho]$ for all rounds r. The proof that $\rho \overset{r}{\sim}_q \rho' \;\Rightarrow\; \sigma \overset{r}{\sim}_q \sigma'$, is by induction on r.

Base case. For the base case, let us consider the initial configuration that corresponds to the fictitious round $r = 0$. Since the initial state of each process (under P as well as under Fip) is fully determined by its initial value, the fact that $\rho \overset{0}{\sim}_q \rho'$ implies that q has the same initial state in both. Since σ corresponds to ρ and σ' corresponds to ρ', it follows that q has the same initial value in σ and σ', and so $\sigma \overset{0}{\sim}_q \sigma'$.

Induction case: $r > 0$. Let us assume that item (i) holds for every process at round $r-1$ (induction assumption). Moreover, let us assume that $\rho \overset{r}{\sim}_q \rho'$. As before, σ and σ' correspond to ρ and ρ', respectively. We need to show that $\sigma \overset{r}{\sim}_q \sigma'$.

Since q survives round r in both the runs ρ and ρ', and as this depends only on their failure patterns F and F', which are also the failure patterns in σ and σ', respectively, we have $q \in S[r,\sigma] \cap S[r,\sigma']$. Let us recall that the local state of a process q at the end of a round r of a run σ of a deterministic algorithm such as P (namely, the local state $ls(q,r,\sigma)$) is a function of its local state in round $r-1$ ($ls(q,r-1,\sigma)$) and the messages that it receives in round r. We have to show that the local states $ls(q,r,\sigma)$ and $ls(q,r,\sigma')$ are the same.

Since $\rho \overset{r}{\sim}_q \rho'$, it follows that the arrays inp_q are the same in both the runs ρ and ρ' at the end of round r. So, $inp_q[q]$ in both runs have the same value at the end of r; let X be that value.

The fact that q survives round r in both ρ and ρ'means, in particular, that it receives its own round r message sent at line 04 of Fip in both ρ and ρ'. The value of this message in ρ is $ls(q,r-1,\rho)$ which is the value of $inp_q[q]$ at the end of r, i.e., $ls(q,r-1,\rho) = X$. A similar reasoning shows that $ls(q,r-1,\rho') = X$. It follows from $ls(q,r-1,\rho) = ls(q,r-1,\rho') = X$ that $\rho \overset{r-1}{\sim}_q \rho'$ and by the inductive hypothesis we obtain $\sigma \overset{r-1}{\sim}_q \sigma'$. Consequently, q has the same local state at the end of round $r-1$ in both σ and σ', i.e., $ls(q,r-1,\sigma) = ls(q,r-1,\sigma')$.

It remains to show that q receives exactly the same messages during round r in both σ and σ'. Suppose that q receives message μ from process $\widehat{q}$ in round r of σ. It follows from the failure pattern F that the message sent by $\widehat{q}$ to q during round r is received by q. Since ρ corresponds to σ and in Fip process $\widehat{q}$ sends messages to all processes in every round, we have that q receives a message from $\widehat{q}$ in round r of ρ as well. As above (case of the message that, at each round, a process sends to itself), this message in ρ contains $ls(\widehat{q},r-1,\rho)$ (the local state of $\widehat{q}$ at round $r-1$ of the run ρ). From $\rho \overset{r}{\sim}_q \rho'$ we have that q receives the same message from $\widehat{q}$ in ρ'. As a result, we have that $\rho \overset{r-1}{\sim}_{\widehat{q}} \rho'$, and by the inductive assumption for $r-1$ and $\widehat{q}$ we obtain that $\sigma \overset{r-1}{\sim}_{\widehat{q}} \sigma'$. Since the message μ is determined by P as a function of the local state of $\widehat{q}$ at $r-1$ in σ, we have that the same message μ is also sent by $\widehat{q}$ to q in round r of σ'. As the round r message sent by $\widehat{q}$ to q in ρ' is received by q, and σ' corresponds to ρ', we obtain that q receives μ from $\widehat{q}$ in σ' as well. It follows that every message received by q in round r of σ is received by it in the same round of σ'. By symmetry, the messages received in σ' are also received in σ. Finally, since q has the same local state in round $r-1$ of σ and σ' (namely, $ls(q,r-1,\sigma) = ls(q,r-1,\sigma')$), and receives the same messages in round r of both σ and σ', we obtain that $\sigma \overset{r}{\sim}_q \sigma'$, which concludes the proof of the lemma. $\square_{Lemma\ 6}$

[4] In the sequel, we interpret $\overset{r}{\sim}$ and $\overset{r}{\approx}$ among runs of an algorithm P in terms of the similarity graph $\mathsf{G}(r,P)$ defined on the runs of P. Thus, the interpretation of $\overset{r}{\sim}$ and $\overset{r}{\approx}$ in statements such as that of Lemma 6 is always with respect to the algorithm generating the related runs.

On failure patterns and full-information algorithms Observe that in both the FIP and PROPOSE algorithms, a correct process is required to send a message to each process in every round. As a result, in runs of both FIP and PROPOSE, a process p knows by the end of round r that q has crashed if the failure pattern F is such that q has crashed before it sent its round r message to p.

The set $f_i[r]$ of the processes that r has not heard from in round r can be directly computed from the local state in FIP as $\{p_j | \, inp_i[j] = \bot\}$. Since p_i sends inp_i to all other processes, the set $f'_i[r-1]$ of processes that p_i knows at r to have been discovered as crashed by round $r-1$ is thus easily computed from the messages it receives in round r. Since for corresponding runs of FIP and PROPOSE these sets coincide (in fact, their values depend only on the failure pattern), we find it convenient to talk about the values of $f_i[r,F]$, $f'_i[r,F]$, $C[r,F]$, $D[F]$, etc. for runs of FIP as well.

Since the Validity property states that it is illegal to decide v in a run that does not contain v as one of its initial values, Corollary 1 implies that it is impossible to decide on v as long as there is a connected run that does not contain v as one of its initial values. In light of this, we can now show that the algorithm FIP reaches a decision as soon as it possibly can.

5.3 Premature rounds

It has been shown in Theorem 2 that the algorithm PROPOSE decides on a value in round $BH = t+1-D$. Let $BH(\rho)$ denote the value of the round number BH of the the run $\rho = \text{PROPOSE}(I,F)$. Recall that the value of BH is solely a function of ρ's failure pattern F (and not of the initial configuration).

Definition 4 *A round ℓ is* premature[5] *in F if $\ell < t+1-D = BH(\rho)$ for every run $\rho = \text{PROPOSE}(I,F)$.*

As $h_i(r+1) = r + (t+1-|f'_i[r]|)$, and $h_i(r+1) \geq BH(\rho)$, this means that, for every r such that $r+1 \leq \ell$, the property $r+t+1-|C[r,F]| > \ell$ holds. Notice that the failure pattern F that occurs during the run ρ determines whether or not ℓ is premature: In all runs of PROPOSE with the same F the sets $f_i[r,F]$ and $C[r,F]$ are the same for every i and r, and so are D and BH.

Lemma 7 *Let $t < n-1$, $\ell \geq 0$, $\rho = \text{FIP}(I,F)$ and $\rho' = \text{FIP}(I,F')$. If $\rho \overset{\ell}{\approx} \rho'$ then ℓ is premature in F iff ℓ is premature in F'.*

Proof Let $\rho = \text{FIP}(I,F)$ and $\rho' = \text{FIP}(I',F')$. As in the proof of item (ii) of Lemma 6, it suffices to show that, for all q, if $\rho \overset{\ell}{\sim}_q \rho'$ then ℓ is premature in F iff ℓ is premature in F'. Thus, let us assume that $\rho \overset{\ell}{\sim}_q \rho'$. Let $\sigma = \text{PROPOSE}(I,F)$ and $\sigma' = \text{PROPOSE}(I',F')$ be the runs of PROPOSE corresponding to the runs $\rho = \text{FIP}(I,F)$ and $\rho' = \text{FIP}(I',F')$, respectively.

By item (ii) of Lemma 6, it follows that $\sigma \overset{\ell}{\sim}_q \sigma'$. Round ℓ is premature in F iff $BH(\sigma) > \ell$, which by Theorem 2 implies that q does not decide in σ by the end of round ℓ. Thus, since the **if** test on line 10 of PROPOSE fails, $\mathsf{decided}_q = \mathit{false}$ continues to hold in σ. The fact that $\sigma \overset{\ell}{\sim}_q \sigma'$ implies that $\mathsf{decided}_q = \mathit{false}$ holds at the end of the round ℓ of σ' as well. It follows that $BH(\sigma') > \ell$, and consequently ℓ is premature at F', as desired. The 'only-if' direction of the lemma is obtained by a symmetric argument. $\Box_{Lemma\ 7}$

Definition 5 *A process is* silent *in a round r of a run ρ if it has crashed before sending its round r messages.*

Definition 6 *Given a failure pattern F, a process q and a round r, let $F_{q,r}$ be the failure pattern that satisfies the following four conditions: (i) $F_{q,r}$ coincides with F for the first $r-1$ rounds, (ii) in round r exactly the failures detected in $C[r,F]$ occur in $F_{q,r}$, (iii) process q is silent from round $r+1$ on, and (iv) no process other than q fails after round r.*

Lemma 8 *If ℓ is premature in F and $k < \ell$, then no more than t processes crash in $F_{q,k}$.*

Proof Let $H[r,F] = r+t+1-|C[r,F]|$. Let us observe that the number of processes that crash in $F_{q,k}$ is at most $|C[k,F]|+1$. It suffices to show that $|C[k,F]| < t$. As ℓ is premature and $k < \ell$, we have $H[k,F] =$

[5] This is short for *premature for simultaneous consensus*, a term that will be justified by the technical analysis in this section.

$k+t+1-|C[k,F]| > \ell$. Since $k < \ell$ we have that $\ell \geq k+1$. We thus obtain that $t+k+1-|C[k,F]| > k+1$, which implies that $t-|C[k,F]| > 0$, and $t \geq |C[k,F]|+1$, as desired.

$\Box_{Lemma\ 8}$

Definition 7 *Given a run $\rho = \textsc{Fip}(I,F)$, let $\rho_{q,k}$ be the run $\rho_{q,k} = \textsc{Fip}(I, F_{q,k})$.*

Let us notice that, due to Lemma 8, if ρ is a run of $\textsc{Fip}$ in which at most t processes fail, then so is $\rho_{q,k}$.

Lemma 9 *Let $t < n-1$ and fix $\ell > 0$. Moreover, let $\rho = \textsc{Fip}(I,F)$ and let $q \in \Pi$. If ℓ is premature in F, then $\rho \overset{\ell}{\approx} \rho_{q,k}$ for all k satisfying $1 \leq k \leq \ell$.*

Proof Let $\ell > 0$. We prove the lemma for all runs ρ, by induction on $d = \ell - k$. For the base case, assume that $d = 0$, and so $k = \ell$. Choose $q \in \Pi$, and let $p \in S[\ell, \rho]$. Since ℓ is premature in ρ, we have $h_p(\ell) = (\ell-1)+t+1-|f_p[\ell-1,F]| > \ell$, i.e., $|f_p[\ell-1,F]| \leq t-1 \leq n-3$. Let $p' \in S[\ell,\rho] \setminus \{p\}$ (such a process p' is guaranteed to exist since $t \leq n-2$).

Let ρ' be the (prefix of a) run identical to ρ up to and including round $\ell-1$, and in round ℓ process p receives the same messages as in ρ, but process p' (who is non-faulty in ρ' too) receives messages from all processes in $\Pi \setminus f_p[\ell-1,F]$. There are exactly $|f_p[\ell-1,F]| < t$ failures in ρ' and $p \in S[\ell,\rho']$ has the same local state at the end of round ℓ in both ρ and ρ'. Hence, we have $\rho \overset{\ell}{\sim}_p \rho'$.

If q is silent in round ℓ in ρ' then we are done. Otherwise, let ρ'' be a run that is the same as ρ' except that q crashes in round ℓ by not sending a round ℓ message to p. At most $|f_p[\ell-1,F]|+1 \leq t-1+1 = t$ processes fail in ρ'', and p' has the same state in ρ' as in ρ''. Hence, $\rho' \overset{\ell}{\sim}_{p'} \rho''$. Finally, observe that $\rho_{q,\ell-1}$ is identical to ρ'' except that q is silent in round ℓ. Process p does not distinguish ρ'' from $\rho_{q,\ell-1}$ since in both it receives the same messages in round ℓ. Thus, $\rho'' \overset{\ell}{\sim}_p \rho_{q,\ell-1}$, and by definition of $\overset{\ell}{\approx}$ we have that $\rho \overset{\ell}{\approx} \rho_{q,\ell-1}$, completing the base case.

Induction step. Let $k < \ell$ and assume that the lemma holds for round $k+1$ in all runs in which ℓ is premature. We prove the lemma for round k. Let ρ be a run in which ℓ is premature, and choose an arbitrary process $q \in \Pi$. We will use the induction assumption to find a connected run that coincides with ρ for the first $k-1$ rounds where no process crashes in round k. If q is silent in this run, then it will be $\rho_{q,k}$ as desired. Otherwise, we use the induction assumption again to show that this run is connected to $\rho_{q,k}$, as desired.

Let ρ' be a run that coincides with ρ for the first $k-1$ rounds where no process crashes in round k, and process p_n is silent from round $k+1$ on. We show that $\rho \overset{\ell}{\approx} \rho'$. Define $\rho^1, \ldots, \rho^n$ where $n = |\Pi|$ to be runs such that in ρ^j the first $k-1$ rounds are identical to ρ, process p_j is silent from round $k+1$ on, no process other than (possibly) p_j fails in rounds $k+1, \ldots, \ell$, and no new failure in round k is seen by processes $p_1, \ldots, p_j$. Denoting $\rho = \rho^0$, we prove by induction on j that $\rho \overset{\ell}{\approx} \rho^j$. The case $j=0$ is immediate, since $\rho^0 = \rho$. Let $j > 0$ and assume inductively that $\rho \overset{\ell}{\approx} \rho^{j-1}$. Since ℓ is premature in $F = F(\rho$, it follows from Lemma 7 that ℓ is premature in ρ^{j-1}. By the induction assumption for $k+1$ we have that $\rho^{j-1} \overset{\ell}{\approx} \widehat{\rho}$, where $\widehat{\rho} = (\rho^{j-1})_{p_j,k}$. In particular, Lemma 8 implies that there are at most t failures in $\widehat{\rho}$. Observe that (i) p_j is silent from round $k+1$ in $\widehat{\rho}$, (ii) every process other than p_j has the same local state at the end of round k in both $\widehat{\rho}$ and ρ^j, and (iii) the same messages are sent from round $k+1$ in both runs, and (iv) since no more processes fail in ρ^j than do in $\widehat{\rho}$, there are at most t failures in ρ^j. It thus follows that $\widehat{\rho} \overset{\ell}{\approx} \rho^j$, completing the induction argument. We conclude that $\rho \overset{\ell}{\approx} \rho^n = \rho'$, as claimed.

Observe that ρ' coincides with ρ for the first $k-1$ rounds and no process crashes in round k. Otherwise, define $\sigma^1, \ldots, \sigma^n$ to be runs of $\textsc{Fip}$ such that in σ^j the first $k-1$ rounds are identical to ρ', process p_j is silent from round $k+1$ on, no process other than (possibly) p_j fails in rounds $k+1, \ldots, \ell$, and round k is identical to ρ' except that process q crashes in round k and does not send messages to $p_1, \ldots, p_j$. Defining $\sigma^0 = \rho'$ an induction argument identical to the one in the previous paragraph shows that $\rho' \overset{\ell}{\approx} \sigma^n$. Notice that q is silent from round k in σ^n. Finally, it follows from the induction hypothesis for $k+1$ that $\sigma^n \overset{\ell}{\approx} \sigma^n_{q,k+1} = \rho_{q,k}$, which completes the proof of the lemma.

$\Box_{Lemma\ 9}$

Based on Lemma 9, we can modify the initial configuration arbitrarily as long as the round number ℓ is premature. Specifically, if there are at least two possible initial values, and all $|V|^n$ initial configurations are possible, then simultaneous agreement on v is impossible: The current run is still connected in $\mathsf{G}(\ell)$ to a run in which all initial values are $\bar{v} \neq v$:[6]

Lemma 10 *Let $t - 1 < n$, and let $v, \bar{v} \in V$ be distinct initial values (i.e., $\bar{v} \neq v$). Let $\rho = \textsc{Fip}(I, F)$ and let ℓ be a premature round in F. There is a run ρ' all of whose initial states are $\bar{v}$, such that $\rho \overset{\ell}{\approx} \rho'$.*

Proof Let $\rho = \textsc{Fip}(I, F)$ and let ℓ be a premature round in F. Thanks to Lemma 9 we can silence the processes one by one from the first round, and change their initial values to $\bar{v}$. Let us then define $\rho^1, \ldots, \rho^n$ to be the runs of $\textsc{Fip}$ such that in $\rho^j = \textsc{Fip}(I^j, F_{p_j,1})$, where I^j coincides with I on the initial values of $p_{j+1}, \ldots, p_n$, and $v_i = \bar{v}$ for all $i \leq j$, and where $F_{p_j,1}$ is obtained from Definition 6 with $q = p_j$ and $r = 1$. Moreover, denote $\rho^0 = \rho$.

We prove by induction on j that $\rho \overset{\ell}{\approx} \rho^j$. For $j = 0$ the claim is trivial, since $\rho = \rho_0$. Assuming that the claim is true for j, we show that it holds for $j + 1$. Since ℓ is premature in F and $\rho \overset{\ell}{\approx} \rho^j$, we have by Lemma 7 that ℓ is premature in $F_{p_j,1}$, which is the failure pattern of ρ^j. Instantiating Definition 7 with $q = p_{j+1}$ and $k = 1$, let us define $\widehat{\rho} = (\rho^j)_{p_{j+1},1}$. By Lemma 9 we have that $\rho^j \overset{\ell}{\approx} \widehat{\rho}$. Notice that $\widehat{\rho}$ differs from ρ^{j+1} only in the initial state of p_{j+1}. Since p_{j+1} is silent in $\widehat{\rho}$ and $t < n$ it follows that $\widehat{\rho} \overset{\ell}{\approx} \rho^{j+1}$. By transitivity of $\approx$ we obtain that $\rho \overset{\ell}{\approx} \rho^{j+1}$, completing the induction step. We finally obtain that $\rho \overset{\ell}{\approx} \rho^n$. Since all initial values in ρ^n are $\bar{v}$, the lemma holds for $\rho' = \rho^n$. $\Box_{Lemma\ 10}$

5.4 Optimality of the proposed algorithm

Lemma 10 combined with Corollary 1 prove that no deterministic algorithm can decide in a run with failure pattern F at a round that is premature in F. Since we have shown in Theorem 2 that Propose is guaranteed to decide in round $BH = BH[F]$ we can conclude that Propose is an optimal algorithm for simultaneous consensus. Hence the following theorem.

Theorem 3 *Let P be a deterministic algorithm for simultaneous consensus. Let σ be a run of P and let ρ be a run of* Propose *corresponding to σ. If the correct processes decide in round r in σ and in round k in ρ, then $k \leq r$.*

6 A few concluding remarks

On the message size of the algorithm Propose Let b be the number of bits required to encode a proposed value. The size of each message is consequently $b + n$ bits. Let us also observe that a process p_i can send a default value $\perp$ instead of est_i when its value is the same as in the previous round. This can reduce the size of some messages when b is big. Another simple improvement to reduce the message size consists in sending, at each round r, the differential value $\Delta f_i[r-1] = f_i[r-1] - f_i[r-2]$ instead of $f_i[r-1]$ ($f_i[-1]$ being initialized to $\emptyset$).

On optimality Usually an algorithm is said to be *optimal* when its performance matches the worst-case lower bound (given a failure model, every execution behaves at least as well as the worst case execution must). Here, the optimality notion is stronger. More precisely, when $t < n - 1$, the proposed algorithm is *optimal* in the sense that, for every particular pattern of failures, no other algorithm can decide in fewer rounds in an execution with the same pattern. So optimality for our algorithm is in each and every run, rather than in the worst case run.

It is shown in [4] that the bound on the minimal number of rounds required for a simultaneous decision is $(t+1)-D$ when $t < n - 1$, and $t - D$ when $t = n - 1$. The algorithm presented in Figure 1, that is optimal when $t < n - 1$, can be easily modified to be optimal also when $t = n - 1$. Essentially, when $t = n - 1$ a process that sees that all others have crashed can immediately decide, and be guaranteed not to violate simultaneity since it is the only process remaining.

[6] In the knowledge-theoretic terminology, the lemma claims that the existence of an initial value of v among the run's initial values is not common knowledge as long as the round is premature.

Circumventing the lower bound An interesting open issue is the following: "Are there additional assumptions that would allow to circumvent the lower bound $RS_{t,F} = (t+1) - D$?" Recall that we have used the fact that all $|V|^n$ initial configurations are possible, as are all ways in which t or less processes can crash. Better performance should be attainable if we assume that the set of initial conditions is more restricted. This is the condition-based approach introduced in [13], and used in [14] to characterize the sets (each set is characterized by a degree d) of the input vectors for which the $\min(t+1, f+2)$ (non-simultaneous) consensus lower bound can be bypassed (it is shown that no more than $\min(t+1, f+2, d+1)$ rounds are necessary when the input vector belongs to a set characterized by the degree d).

References

[1] Attiya H. and Welch J., *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd Edition)*, Wiley Interscience, 414 pages, 2004.

[2] Dolev D., Reischuk R. and Strong R., Early Stopping in Byzantine Agreement. *Journal of the ACM*, 37(4):720-741, April 1990.

[3] Dwork C. and Moses Y., Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures. *Information and Computation*, 88(2):156-186, 1990.

[4] Fagin R., Halpern J.Y., Moses Y. and Vardi M.Y., *Reasoning about Knowledge*, MIT Press, 533 pages, 2003.

[5] Fischer M.J. and Lynch N., A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 71:183-186, 1982.

[6] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[7] Garg V.K., *Elements of Distributed Computing*, Wiley, 423 pages, 2002.

[8] Halpern J.Y. and Moses Y., Knowledge and Common Knowledge in a Distributed Environment. *Journal of the ACM*, 37(3):549-587, 1990.

[9] Lynch N.A., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Francisco (CA), 872 pages, 1996.

[10] Mizrahi T. and Moses Y., Continuous Consensus via Common Knowledge. *Distributed Computing*, 20(5):305-321, 2008.

[11] Mizrahi T. and Moses Y., Continuous Consensus with Failures and Recoveries. *Unpublished manuscript*, 2008.

[12] Moses Y. and Rajsbaum S., A Layered Analysis of Consensus. *SIAM Journal of Computing*, 31(4):989-1021, 2002.

[13] Mostéfaoui A., Rajsbaum S. and Raynal M., Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems. *Journal of the ACM*, 50(6):922-954, 2003.

[14] Mostéfaoui A., Rajsbaum S. and Raynal M., Synchronous Condition-Based Consensus. *Distributed Computing*, 18(5):325-343, 2006.

[15] Pease L., Shostak R. and Lamport L., Reaching Agreement in Presence of Faults. *Journal of the ACM*, 27(2):228-234, 1980.

[16] Raynal M., Consensus in Synchronous Systems: a Concise Guided Tour. *Proc. 9th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'02)*, IEEE Computer Press, pp. 221-228, 2002.

[17] Wang X., Teo Y.M. and Cao J., A Bivalency Proof of the Lower bound for Uniform Consensus. *Information Processing Letters*, 96:167-174, 2005.