Tight Bounds on Early Local Decisions in Uniform Consensus (Extended Abstract)*

Partha Dutta, Rachid Guerraoui, and Bastian Pochon

Distributed Programming Laboratory, EPFL

Abstract. When devising a uniform consensus algorithm, it is common to minimize the time complexity of *global* decisions, which is typically measured as the number of communication rounds needed for *all* correct processes to decide. In practice, what we might want to minimize is the time complexity of *local* decisions, which we define as the number of communication rounds needed for *at least one* correct process to decide. We investigate tight bounds on uniform consensus local decisions in crash-stop message-passing models where at most t processes may fail in any given run.

In the synchronous model, we show that any uniform consensus algorithm has (1) a run in which at most $f \leq t-1$ processes crash such that *no correct process* decides before round $f+1$ in that run, and (2) a run in which at most $f \leq t-3$ processes crash such that *at most one correct process* decides before round $f+2$ in that run. We show that the above lower bounds are tight by pointing out a simple uniform consensus algorithm.

In the eventually synchronous model, we show that any uniform consensus algorithm has a *synchronous run* in which at most $f \leq t-3$ processes crash such that *no correct process* decides before round $f+2$ in that run. We present a new uniform consensus algorithm that globally decides in $f+2$ rounds in every synchronous run in which at most f processes crash. Thus the local and the global decision tight bounds are the same for synchronous runs of the eventually synchronous model.

1 Introduction

Motivation. Determining how long it takes to reach uniform consensus [11, 16] among a set of processes is an important question in distributed computing. For instance, the performance of a replicated system is impacted by the performance of the underlying uniform consensus service used to ensure that the replica processes agree on the same order to deliver client requests [15]. Traditionally, lower bounds on the time complexity of uniform consensus have been stated in terms of the number of communication rounds (also called communication steps) needed

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

for *all* correct processes to decide [3,8,13] (i.e., *global decision*), or even *halt* [4], possibly as a function of the number of failures f that actually occur, out of the total number t of failures that are tolerated.

From a practical perspective, what we might sometimes want to measure and optimize, is the number of rounds needed for *at least one* correct process to decide, i.e., for a *local decision*. Indeed, a replicated service can respond to its clients as soon as a single replica decides on a reply and knows that other replicas will reach the same decision (even if they did not decide yet).

Background. We consider two message-passing crash-stop models where communication among a set of n processes proceed by exchanging messages in a round by round manner: (1) the well-known synchronous model [14] and (2) an eventually synchronous model. In any run, at most t processes might fail, and they can only do so by crashing. In a given run, if a process crashes in that run, it is called faulty; otherwise, it is correct.

In the synchronous model, the $f+2$ round global decision lower bound states the following: for every $f \leq t-2$, any uniform consensus algorithm has a run in which at most f processes crash such that some correct process decides in round $f+2$ or in a higher round in that run [3, 13]. In other words, for all correct processes to decide, we need at least $f+2$ rounds. However, a global decision lower bound does not say whether *some* correct process can decide before round $f+2$ in every run with f crashes, and if yes, how many processes may actually do so.[1]

In the eventually synchronous model, we know from [9] that for every $f \leq t$, any uniform consensus algorithm has a run in which at most f processes crash such that processes decide after arbitrarily large number of rounds in that run (because the run may remain "asynchronous" for an arbitrary number of rounds). However, if we focus on the *synchronous runs* of any algorithm (i.e., the runs which are "synchronous" from the very beginning) we can bound the number of rounds required for processes to decide. In fact it is easy to see that the $f+2$ round global decision lower bound in synchronous model immediately extends to synchronous runs of the algorithms in the eventually synchronous model. However, unlike the synchronous model, devising an algorithm that matches the $f+2$ round global decision lower bound has been an open problem [5, 12].

Contributions. This paper first shows that, in the synchronous model, the local decision tight bound is $f+1$. In other words, for every $f \leq t-1$, (1) any uniform consensus algorithm has a run in which at most f processes crash such that no correct process decides in a round lower than $f+1$ in that run, and (2) there is a uniform consensus algorithm such that in every run in which at most f processes crash, some correct process decides by round $f+1$.

Furthermore, we show that, for $f \leq t-3$, any uniform consensus algorithm has a run in which at most f processes crash such that either none or exactly one process decides before round $f+2$ in that run; this gives a bound on the

[1] We assume that processes may take steps after their decision events.

number of correct processes which can decide before the global decision lower bound (i.e., $f+2$) in every run with at most f failures. This result generalizes the global decision lower bound of [3, 13] which states that any uniform consensus algorithm has a run in which at most f processes crash such that at least one correct process decides in round $f+2$ or in a higher round in that run: our result implies that there is a run in which at most f processes crash such that at least $n-f-1$ correct processes decide in round $f+2$ or in a higher round in that run (because there are at least $n-f$ correct processes in a run in which at most f processes crash, and at most one of them can decide before round $f+2$).

In the eventually synchronous model, we show that, for $f \leq t-3$, any uniform consensus algorithm has a synchronous run in which at most f processes crash such that no process decides before round $f+2$ in that run. We give a matching algorithm which globally decides (and hence, locally decides) by round $f+2$ in every synchronous run in which at most f processes crash, for every $0 \leq f \leq t$. Thus the local decision tight bound is the same as the global decision tight bound.

Roadmap. Section 2 presents the two distributed system models we consider. Section 3 recalls the uniform consensus problem and its time complexity metrics. Section 4 presents a key lemma for our proofs, which is a variant of a lemma of [13]. Section 5 presents our lower bound proofs in the synchronous model and Section 6 presents the lower bound proof in the eventually synchronous model. Finally, Section 7 exhibits a matching algorithm for the eventually synchronous model.

2 Models

We assume a distributed system composed of $n \geq 3$ processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$. The processes communicate by message-passing and every pair of processes is connected by a bi-directional communication channel. Channels do not alter or duplicate messages. The processes may fail by crashing and do not recover from a crash. Any process that does not crash in a run is said to be correct in that run; otherwise the process is faulty. In any given run, at most $t < n$ processes may crash. We consider two round-based models: the well-known synchronous model [14], which we denote by *SCS*, and an eventually synchronous model, which we denote by *ES*. Computation proceeds in rounds of message exchanges with increasing round numbers starting from 1. Each round consists of a two phases: (a) in the *send phase*, processes are supposed to send messages to all processes; (b) in the *receive phase*, processes receive some messages sent in the send phase, update local states, and (possibly) decide. We say that a message m sent from p_i to p_j is *lost*, if p_j never receives m in that run. We now describe the guarantees of the two models we consider:

- In *SCS*, if a process p_i crashes in the send phase of some round k, then any subset of the messages sent by p_i in that round may be lost, but the remaining round k messages sent by p_i are received in round k. If p_i does not crash in the send phase of round k, then no process completes round k without receiving the round k message from p_i.

- In *ES*, the runs may be "asynchronous" for an arbitrary yet finite number of rounds but eventually become "synchronous." A message sent in the "asynchronous period" may be *delayed* for a finite number of rounds; i.e., received in a round higher than in which it was sent. More precisely, for every run in *ES*, the following properties hold: (1) (*t-resilience*) Every process which completes any round k, receives the round k messages from at least $n-t$ processes, (2) (*reliable channels*) any message sent by a correct process to a correct process is never lost but may be delayed for an arbitrary yet finite number of rounds, (3) (*eventual synchrony*) there is an unknown but finite round number K such that, in every round $k \geq K$, if p_i crashes in round k then any subset the messages sent by p_i in that round may be lost, but the remaining round k messages sent by p_i are received in round k, and if p_i does not crash in round k, then no process completes round k without receiving the round k message from p_i.

We say that a run in *ES* is *synchronous* if $K = 1$ in that run. Obviously, for every run $r1$ in *SCS* there is a synchronous run $r2$ in *ES* such that no process can distinguish $r1$ from $r2$.

3 Time Complexity of Uniform Consensus

We define the uniform consensus problem using two primitives: propose($*$) and decide($*$). Every process is supposed to propose a value through the procedure propose($*$), and a process decides a value by invoking decide($*$). Uniform consensus ensures the following properties: (1) (*validity*) if a process decides v then some process has proposed v, (2) (*uniform agreement*) no two processes decide differently, and (3) (*termination*) every correct process eventually decides. Binary uniform consensus is a variant of uniform consensus in which proposal values are restricted to 0 and 1. We only consider deterministic uniform consensus algorithms.

Consider any uniform consensus algorithm in *SCS* or *ES*. We say that a process decides in round $k \geq 1$ if it decides in the receive phase of round k. A run of an algorithm *globally decides* in round k if all correct processes decide in round k or in a *lower* round, and some correct process decides in round k. A run of an algorithm *locally decides* in round k if all correct processes decide in round k or in a *higher* round and some correct process decides in round k.

In *SCS* (resp. *ES*), for every $0 \leq f \leq t$, we define the *global decision tight bound* as the round number k such that: (1) every uniform consensus algorithm has a run (resp. synchronous run) in which at most f processes crash and which globally decides in round k or in a higher round, and (2) there is a uniform consensus algorithm, which globally decides by round k in every run (resp. synchronous run) in which at most f processes crash. For every $0 \leq f \leq t$, we define the *local decision tight bound* as the round number k such that: (1) every uniform consensus algorithm has a run (resp. synchronous run) in which at most f processes crash and which locally decides in round k or in a higher round, and (2) there is a uniform consensus algorithm, which locally decides by round k in every run (resp. synchronous run) in which at most f processes crash.

4 Proof Preliminaries

Our lower bound proofs are devised following the layering technique of [17], also used in [13]. Since every uniform consensus algorithm also solves the binary version of the problem, for lower bound proofs, we consider binary uniform consensus algorithms only.

Consider any algorithm A that solves binary uniform consensus in *SCS*. A *round k configuration* ($k \geq 1$) in a run of A consists of the states of all processes at the end of round k. (Since in *SCS*, no message is received in a round different from the one in which it was sent, we do not need to consider the states of the communication channels at the end of a round.) The state of a process which has crashed in a configuration is a special symbol denoting that the process has crashed. We say that a process p_i is *alive* in a given configuration if p_i has not crashed in that configuration. An *initial configuration* (or round 0 configuration) in a run consists of initial states of all processes in that run. For any configuration C at round k, we define $r(C)$ as the run in which (1) round k configuration is C, and (2) no processes crashes after round k. We denote by $val(C)$ the decision value of the correct processes in $r(C)$. Note that a process p_i is alive in C iff p_i is correct in $r(C)$.

Two configurations C and D at the same round are *similar*, denoted $C \sim D$, if they are identical or there exists a process p_j such that (1) C and D are identical except at p_j, and (2) there exists a process $p_i \neq p_j$ that is alive in both C and D. A set of configurations SC is *similarity connected* if, for every $C, D \in SC$ there are configurations $C = C_0, \ldots, C_m = D$ such that $C_i \sim C_{i+1}$ for all $0 \leq i < m$.

Let $scs1(A)$ be the runs of A in *SCS* in which the following holds: (1) at most one process crashes in every round, and (2) in any round k, if some process p_i crashes then: if some process p_j does not receive the round k message from p_i then every process p_l such that $l < j$ does not receive the round k message from p_i. (In other words, messages from the crashed process are lost by only a prefix of Π.) Let $Init$ be the set of initial configurations of all the runs in $scs1(A)$.

For the runs in $scs1(A)$, [13] shows the following two lemmas. Lemma 2.1 of [13] states that, if $n > 1$, then $Init$ is similarity connected. Lemma 2.3 of [13] states that: for $k \leq t$, if $Init$ is similarity connected, then the set of round k configurations of all runs in $scs1(A)$ is a similarity connected set of configurations in which no more than k processes have crashed in each configuration. Using these lemmas, we now sketch the proof of the following lemma. (We present a detailed proof in the full-version of the paper [6].)

Lemma 1. *For any k such that $0 \leq k \leq t$, there are two runs of A in* SCS *such that their round k configurations, y and y', satisfy the following: (1) at most k processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $val(y) = 0$, whereas $val(y') = 1$.*

Proof. (Sketch) Since every run in $scs1(A)$ is a run of A in *SCS*, it is sufficient to show that there are two round k configurations of the runs in $scs1(A)$ with the above properties.

Consider a run $r0$ of A in which every process proposes 0 and no process crashes. From the definition of $scs1(A)$, $r0$ is in $scs1(A)$. Consider the round k configuration, say y, of $r0$. From uniform consensus validity, $val(y) = 0$. Similarly, consider a run $r1$ in which every process proposes 1 and no process crashes. Clearly, $r1$ is in $scs1(A)$. Consider the round k configuration, say y' of $r1$. From uniform consensus validity, $val(y') = 1$.

Since we assume $n \geq 3$ and $k \leq t$, Lemma 2.3 of [13] implies that, the set of round k configurations of all runs in $scs1(A)$ is a similarity connected set of configurations in which no more k processes have crashed in each configuration. From the definition of similarity connected, it follows that there are some round k configurations in $scs1(A)$, $y = y_0, y_1, \ldots, y_m = y'$, such that $y_j \sim y_{j+1}$ for all $0 \leq j < m$. Clearly, there is some $y_i \in \{y_0, \ldots, y_{m-1}\}$ such that, $val(y_0) = \ldots = val(y_i) \neq val(y_{i+1})$. (Otherwise, $val(y) = val(y_0) = val(y_1) = \ldots = val(y_m) = val(y')$, a contradiction.)

Thus, $val(y_i) = 0$ and $val(y_{i+1}) = 1$. As $y_i \sim y_{i+1}$, the two configurations are either identical or differ at exactly one processes. Since, $val(y_i) \neq val(y_{i+1})$, the configurations cannot be identical.

5 Lower Bounds in Synchronous Model

Proposition 1. *Let $1 \leq t \leq n-1$. For every f such that $0 \leq f \leq t-1$, any uniform consensus algorithm has a run in which at most f processes crash such that no correct process decides before round $f+1$ in that run.*

Proof. Suppose by contradiction that there is an algorithm A that solves uniform consensus in *SCS*, a round number f such that $0 \leq f \leq t-1$, and in every in which at most f processes crash, some correct process decides before round $f+1$.

From Lemma 1 we know that there are two runs of A in *SCS* such that their round f configurations, y and y', satisfy the following: (1) at most f processes have crashed in each configuration, (2) the configurations differ at exactly one process, say p_i, and (3) $val(y) = 0$ and $val(y') = 1$.

From our initial assumption, it follows that in y there is an alive process q_1 which has already decided. (Otherwise, since every correct process in $r(y)$ is an alive process in y, $r(y)$ is a run with f crashes in which no correct process decides before round $f+1$.) Furthermore, q_1 has decided $val(y) = 0$ in y because q_1 is a correct process in $r(y)$. Similarly, in y', there is an alive process q_2 which has decided $val(y') = 1$. There are two cases to consider.

(1) $q_1 \neq p_i$: As y and y' are identical at all processes different from p_i, in y', q_1 is alive and has decided 0. Thus in $r(y')$, q_1 is a correct process and decides 0. However, in $r(y')$ every correct process decides $val(y') = 1$; a contradiction.

(2) $q_1 = p_i$: We distinguish two subcases:

- $q_2 = p_i$: Thus $p_i = q_1 = q_2$, and hence, p_i is alive in y and y'. Consider a run $r1$ which extends y and in which p_i crashes before sending any message in round $f+1$. (Recall that $f \leq t-1$.) As p_i has decided 0 in y, from uniform agreement, it follows that every correct process decides 0 in $r1$. Since $t < n$, there is at least one correct process, say p_l in $r1$. Now consider a run $r2$ which extends y' and in which p_i crashes before sending any message in round $f+1$. Notice that no correct process can distinguish $r1$ from $r2$: at the end of round f no alive process which is distinct from p_i can distinguish y from y', and p_i crashes before sending any message in round $f+1$. Thus every correct process decides the same value in $r1$ and $r2$, in particular p_l decides 0 in $r2$. However, $p_i = q_2$ decides 1 in $r2$; a contradiction with uniform agreement.
- $q_2 \neq p_i$: Then, q_2 has the same state in y and y'. Thus in y, q_2 is alive and has decided 1. In any run which extends y, $p_i = q_1$ has decided 0 and q_2 has decided 1; a contradiction with uniform agreement. □

It is easy to design an algorithm that matches Proposition 1. We give a brief sketch of one such algorithm. Every process maintains a variable *est* which is initialized to the proposal value of the process. In each round i, such that $1 \leq i \leq t+1$, process p_i sends its *est* to all processes, and if any process p_j receives the round i message from p_i, p_j updates its own *est* to the *est* received from p_i. At the end of round $i \leq t$, p_i decides on its own *est* value, and at the end of round $t+1$, every alive process which has not yet decided, decides on its own *est* value. Validity and termination properties of the algorithm are easy to see. To see why the agreement property holds, consider the first process p_i which decides. If $i \leq t$, then p_i completed the send phase of round i, and hence, every process which completes round i adopts the *est* value of p_i. Thus, no value distinct from the *est* sent by p_i in round i can be decided in a higher round. If $i = t+1$, then every process in $\{p_1, \ldots, p_t\}$ are faulty, and hence, p_{t+1} is correct. Thus, every process receives the *est* of p_{t+1} in round $t+1$, and decides on that value. To see the early local decision property of the algorithm, notice that in a run in which at most f processes crash, at least one of the the processes in $\{p_1, \ldots, p_{f+1}\}$ is correct, and it decides by the end of round $f+1$. In the full-version of the paper [6], we further optimize the algorithm for early global decision and discuss the special case for $f = t$.

From the above tight bound on local decision, we know that, in runs in which at most f processes crash, the decision of any correct process requires $f+1$ rounds. On the other hand, the global decision lower bound states that, in runs in which at most f processes crash, decision of all correct processes requires $f+2$ rounds. It is natural to ask whether we can devise a uniform consensus algorithm in which more than one correct process can decide in round $f+1$, in every run in which at most f processes crash. In the following proposition we show that the answer is negative.

Proposition 2. *Let $3 \leq t \leq n-1$. For every f such that $0 \leq f \leq t-3$, any uniform consensus algorithm has a run in which at most f processes crash such that at most one correct process decides before round $f+2$ in that run.*

Proof. Suppose by contradiction that there is an algorithm A that solves uniform consensus in SCS, a round number $f+1$ such that $0 \leq f \leq t-3$, and in every run of A in which at most f processes crash, there are two correct processes which decide before round $f+2$.

From Lemma 1 we know that there are two runs of A in SCS such that their round f configurations, y and y', satisfy the following: (1) at most f processes have crashed in each configuration, (2) the configurations differ at exactly one process, say p_i, and (3) $val(y) = 0$ and $val(y') = 1$. Let z and z' denote the configurations at the end of round $f+1$ of $r(y)$ and $r(y')$, respectively.

From our initial assumption, it follows that in z, there are two alive processes q_1 and q_2 which have decided $val(y) = 0$. Similarly, in z', there are two alive processes q_3 and q_4 which have decided $val(y') = 1$. Since q_1 and q_2 are distinct, at least one of them is distinct from p_i, say q_1. Similarly, without loss of generality we may assume that q_3 is distinct from p_i.

Thus we have (1) an $f+1$ round configuration z in which at most f processes have crashed such that z has an alive process q_1 which has decided 0, (2) an $f+1$ round configuration z' in which at most f processes have crashed such that z' has alive process q_3 which has decided 1, and (3) process p_i is distinct from both q_1 and q_3. (Processes q_1 and q_3 may or may not be distinct.) There are two cases to consider.

Case 1. Process p_i is alive in y and y'. Consider the following two runs of A in SCS:
R1 is a run such that (1) the round f configuration is y, (2) p_i crashes in the send phase of round $f+1$ such that only q_1 receives the message from p_i, (3) q_1 and q_3 crash before sending any message in round $f+2$, and (4) no process distinct from p_i, q_1, and q_3 crashes after round f. Notice that q_1 cannot distinguish the round $f+1$ configuration of $R1$ from z, and therefore, decides 0 at the end of round $f+1$ in $R1$. By uniform agreement, every correct process decides 0. Since $t \leq n-1$, there is at least one correct process in $R1$, say p_l.
R2 is a run such that (1) the round f configuration is y', (2) p_i crashes in the send phase of round $f+1$ such that only q_3 receives the message from p_i, (3) q_1 and q_3 crash before sending any message in round $f+2$, and (4) no process distinct from p_i, q_1, and q_3 crashes after round f. Notice that q_3 cannot distinguish the round $f+1$ configuration of $R2$ from z', and therefore, decides 1 at the end of round $f+1$ in $R2$. However, p_l cannot distinguish $R1$ from $R2$: at the end of round $f+1$, the two runs are different only at p_i, q_1, and q_3, and none of the three processes sends messages after round $f+1$ in both runs. Thus (as in $R1$) p_l decides 0 in $R2$; a contradiction with uniform agreement.

Case 2. Process p_i has crashed in either y or y'. (Process p_i is not crashed in both y and y' because p_i has different states in y and y'.) Without loss of generality, we can assume that p_i has crashed in y, and hence, p_i is alive in y'. (Recall that the state of p_i in y is distinct from that in y'.) Consider the following two runs of A in SCS:

R12 is a run such that (1) the round f configuration is y (and hence, p_i has crashed before round $f+1$), (2) no process crashes in round $f+1$, (3) q_1 and q_3 crash before sending any message in round $f+2$, and (4) no process distinct from p_i, q_1 and q_3 crashes after round f. Observe that the round $f+1$ configuration of $R12$ is z, and hence, q_1 decides 0 at the end of round $f+1$ in $R12$. Due to uniform agreement, every correct process decides 0 in $R12$. Since $t \leq n-1$, there is at least one correct process in $R12$, say p_l.

R21 is a run such that (1) the round f configuration is y', (2) p_i crashes in the send phase of round $f+1$ such that only q_3 receives the message from p_i, (3) q_1 and q_3 crash before sending any message in round $f+2$, and (4) no process distinct from p_i, q_1 and q_3 crashes after round f. Notice that q_3 cannot distinguish the round $f+1$ configuration of $R21$ from z' because it receives the message from p_i in both runs. Thus (as in z') q_3 decides 1 at the end of round $f+1$ in $R21$. However, p_l cannot distinguish $R12$ from $R21$: at the end of round $f+1$, the two runs are different only at p_i, q_1 and q_3, and none of them sends messages after round $f+1$ in both runs. Thus (as in $R12$), p_l decides 0 in $R21$; a contradiction with uniform agreement. □

6 Lower Bound in Eventually Synchronous Model

In synchronous runs of any uniform consensus algorithm in ES we show that there is a run in which at most f processes crash, such that no correct process decides before round $f+2$ in that run; i.e., the local decision lower bound is $f+2$. This one round difference between local decisions in SCS and that of synchronous runs in ES, can be seen as a price paid by algorithms in ES to tolerate a weaker model.

In ES, a *round k configuration* ($k \geq 1$) consists of the states of all processes at the end of round k and the set of delayed messages in each communication channel at the end of round k. (Since messages may be delayed, to completely describe the state of the system at the end of a round, we need to take into account the delayed messages in every channel.) An initial configuration (or round 0 configuration) in a run consists of initial states of all processes in that run and empty communication channels. A *synchronous configuration* is a configuration of some synchronous run. As there are no delayed messages in a synchronous run, all communication channels are empty in any synchronous configuration. For any synchronous configuration C at round k, we define $R(C)$ as a synchronous run in which (1) round k configuration is C, and (2) no processes crashes after round k. We denote by $Val(C)$ the decision value of the correct processes in $R(C)$.

Consider any algorithm A that solves uniform consensus in ES. As no process can distinguish a run of A in SCS from some synchronous run of A in ES, and the communication channels are empty in every synchronous configuration, it follows from Lemma 1 that,

Claim ES1. For any k such that $0 \le k \le t$, there are two distinct synchronous runs of A in ES such that their round k synchronous configurations, y and y', satisfy the following: (1) at most k processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $Val(y) = 0$, whereas $Val(y') = 1$.

Proposition 3. *Let $1 \le t \le n-1$. For every f such that $0 \le f \le t-3$, any uniform consensus algorithm has a synchronous run in which at most f processes crash such that no correct process decides before round $f+2$ in that run.*

Proof. Suppose by contradiction that there is an algorithm A that solves uniform consensus in *ES*, a round number f such that $0 \le f \le t-3$, and in every synchronous run of A in which at most f processes crash, some correct process decides by round $f+1$. From Claim *ES1*, we know that there are two round f synchronous configurations of A, y and y', such that: (1) at most f processes have crashed in each configuration, (2) the configurations differ at exactly one process, say p_i, and (3) $Val(y) = 0$ and $Val(y') = 1$. Let z and z' denote the round $f+1$ configurations in the synchronous runs $R(y)$ and $R(y')$, respectively.

From our initial assumption on A, in z, there is at least one alive process, say q_1, which has decided 0. Similarly, in z', there is at least one alive process, say q_3, which has decided 1. There are three cases to consider.

Case 1. $p_i \notin \{q_1, q_3\}$. This case is exactly similar to the case in the proof of Proposition 2. We can derive a contradiction by constructing the same runs R1, R2, R12, and R21. (We can ignore the states of the communication channels in any synchronous configuration because the channels are empty in such a configuration.)

Case 2. $p_i \in \{q_1, q_3\}$ and p_i is alive in both y and y'. Notice that if $p_i = q_1$ then $R1$ (in the proof of Proposition 2) is not a run of A ES: p_i cannot crash in the send phase of round $f+1$, and decide at the end of round $f+1$. (Similarly, if $p_i = q_3$ then $R2$ is not a run in *ES*.) Hence we construct non-synchronous runs of A to show the contradiction. Without loss of generality we can assume that $p_i = q_1$. (Note that the proof holds even if $p_i = q_1 = q_3$.) Consider the following synchronous run $R3$ and two *non-synchronous* runs, $R4$ and $R5$.

R3 is a run such that (1) the round f configuration is y, (2) p_i crashes in round $f+1$ before sending any message, (3) if $q_3 \ne p_i$ then q_3 crashes before sending any message in round $f+2$ and every message sent by q_3 in round $f+1$ is received in round $f+1$, and (4) no process distinct from p_i and q_3 crashes after round f. Since $t \le n-1$, there is at least one correct process in $R3$, say p_l. Suppose p_l decides $v \in \{0,1\}$ in some round $K1 \ge f+1$.[2]

[2] To see that p_l cannot decide before round $f+1$ in $R3$, notice that the state of p_l at the end of round f is the same in runs $R(y)$, $R(y')$ and $R3$. If p_l decides v before round $f+1$ in $R3$ then it also decides v in $R(y)$ and $R(y')$. However, $Val(y) \ne Val(y')$.

R4 is a run such that (1) the round f configuration is y, (2) in round $f+1$, p_i and q_3 receive the round $f+1$ message from p_i, and the round $f+1$ message from p_i to other processes are delayed until round $K1+1$, (3) p_i and q_3 crashes in round $f+2$ before sending any message in that round, and (4) no process distinct from p_i and q_3 crashes after round f. Notice that p_i cannot distinguish the configuration at the end of round $f+1$ in $R4$ from z, and thus, p_i decides 0 at the end of round $f+1$ in $R4$. However, p_l cannot distinguish the round $K1$ configuration of $R4$ from that of $R3$ because (1) at the end of round f, the two runs are different only at p_i, and all round $f+1$ messages from p_i to processes distinct from p_i and q_3 are delayed until round $K1+1$, and (2) p_i and q_3 do not send messages after round $f+1$. Thus (as in $R3$) p_l decides v at the end of round $K1$.

R5 is a run such that (1) the round f configuration is y', (2) in round $f+1$, p_i and q_3 receive the round $f+1$ message from p_i, and the round $f+1$ message from p_i to other processes are delayed until round $K1+1$, (3) p_i and q_3 crashes in round $f+2$ before sending any message in that round, and (4) no process distinct from p_i and q_3 crashes after round f. Notice that q_3 cannot distinguish the configuration at the end of round $f+1$ in $R5$ from z' (because q_3 receives the message from p_i in round $f+1$ of $R5$), and thus, q_3 decides 1 at the end of round $f+1$ in $R5$. However, p_l cannot distinguish the round $K1$ configuration of $R5$ from that of $R3$ because, (1) at the end of round f the two runs are different only at p_i, and all round $f+1$ messages from p_i to processes distinct from p_i and q_3 are delayed until round $K1+1$, and (2) p_i and q_3 do not send messages after round $f+1$. Thus (as in $R3$) p_l decides v at the end of round $K1$.

Clearly, either $R4$ or $R5$ violates uniform agreement: p_l decides v in both runs, however, p_i decides 0 in $R4$ and q_3 decides 1 in $R5$.

Case 3. $p_i \in \{q_1, q_3\}$ and p_i has crashed in either y or y'. (Process p_i is not crashed in both y and y' because p_i has different states in y and y'.) Notice that the case $p_i = q_1 = q_3$ is not possible because, in that case, p_i is alive in z and z', and hence in y and y'. We show the contradiction for the case when $p_i = q_1 \neq q_3$. (The contradiction for $p_i = q_3 \neq q_1$ is symmetric.)

Since, $p_i = q_1$, p_i is alive in z, and hence, alive in y. Thus p_i has crashed in y'. Consider the following non-synchronous run.

R6 is a run such that (1) the round f configuration is y, (2) in round $f+1$, only p_i receives the round $f+1$ message from itself, and the round $f+1$ message from p_i to other processes are delayed until round $f+2$, (3) p_i crashes in round $f+2$ before sending any message in that round, and (4) no process distinct from p_i crashes after round f. At the end of round $f+1$ in $R6$, $p_i = q_1$ cannot distinguish the configuration from z, and therefore, decides 0. However, q_3 does not receive the round $f+1$ message from p_i in $R6$ (the message is delayed until round $f+2$), and hence, q_3 cannot distinguish the configuration at the end of round $f+1$ in $R6$ from z'. (In z', q_3 does not receive the round $f+1$ message from p_i because p_i has crashed in y'.) Consequently, q_3 decides 1 in $R6$; a contradiction with uniform agreement. $\square$

A closer look at the proof of Proposition 3 reveals that the non-synchronous runs we construct (R4, R5, and R6) satisfy the following "weak synchrony" property: *for any round k and any process p_i, if a round k message from p_i is delayed, then p_i crashes before sending any message in round $k+1$.* It is easy to see that such runs can also be constructed in the synchronous send-omission model [10] as well as in an asynchronous round-by-round model enriched with a *Perfect* failure detector [2]. Thus the $f+2$ local decision lower bound in synchronous runs also extend to these two models.

7 A Matching Algorithm

Figure 1 presents a uniform consensus algorithm A_{f+2} in *ES* which matches the $f+2$ round global decision lower bound (and hence, matches the local decision bound) in synchronous runs. Namely, the algorithm satisfies the following property: (*Fast Early Decision*) For $0 \leq t < n/2$,[3] in every synchronous run of A_{f+2} in which at most f processes crash ($0 \leq f \leq t$), every process which decides, decides by round $f+2$.

For simplicity of presentation, A_{f+2} assumes an independent uniform consensus algorithm C,[4] accessed through procedure propose$_C(*)$. The fast decision property is achieved by A_{f+2} regardless of the time complexity of C. More precisely, our algorithm assumes: (1) the model *ES* with $0 \leq t < n/2$, (2) messages sent by a process to itself are received in the same round in which they are sent, (3) an independent uniform consensus algorithm C in *ES*, and (4) the set of proposal values in a run is a totally ordered set, e.g., every process p_i can tag its proposal value with its index i and then the values can be ordered based on this tag.

The processes invoke propose($*$) with their respective proposal values, and the procedure progresses in rounds. Every process p_i maintains three primary variables:

- STATE$_i$ is updated at the end of a round as follows. If p_i considers (a) the run to be non-synchronous then p_i sets STATE$_i$ to NSYNC, (b) the run to be synchronous but p_i cannot decide at the next round then p_i sets STATE$_i$ to SYNC1, (c) the run to be synchronous with a possibility of deciding at the next round then p_i sets STATE$_i$ to SYNC2.
- est_i is the estimate of the possible decision value, and roughly speaking, the minimum value seen by p_i.
- $Halt_i$ is a set of processes. At the end of a round, $Halt_i$ contains p_j if any of the following holds in the current round or in a lower round: (1) p_i did not receive a message from p_j, (2) p_i receives a messages from p_j with STATE $=$ NSYNC, or (3) p_i receives a messages from p_j containing $Halt_j$ such that $p_i \in Halt_j$.

[3] Uniform consensus is impossible to solve in *ES* when $t \geq n/2$ [2].

[4] This algorithm can be any $\Diamond P$-based or $\Diamond S$-based uniform consensus algorithm (e.g., the one based on $\Diamond S$ in [2]) transposed to *ES*.

at process p_i

1: **procedure** propose(v_i)
2: **start Task1; start Task 2**

3: **Task 1**
4: $\text{STATE}_i \leftarrow$ SYNC1 ; $est_i \leftarrow v_i$; $Halt_i \leftarrow \emptyset$
5: **for** $1 \le k_i \le t+2$
6: send(k_i, est_i, STATE_i, $Halt_i$) to all
7: **wait until** received messages of round k_i
8: **if** received any (k_i, est', DECIDE, $*$) **then**
9: decide(est'); send(k_i+1, est', DECIDE, $\emptyset$) to $\Pi \backslash p_i$; **return** $\{decision\}$
10: **if** $\text{STATE}_i \in$ {SYNC1, SYNC2} **then**
11: $Halt_i \leftarrow Halt_i \cup \{p_j \mid (p_i$ received(k_i, $*$, NSYNC, $*$) from p_j) **or** (p_i received(k_i, $*$, $*$, $Halt_j$) from p_j s.t. $p_i \in Halt_j$) **or** (p_i did not receive any round k_i message from p_j)}
12: $msgSet_i \leftarrow \{\, m \mid m$ is a round k_i message received from $p_j \notin Halt_i\}$
13: $est_i \leftarrow$ **Min**$\{est \mid (k_i, est, *, *) \in msgSet_i\}$
14: **if** ($\text{STATE}_i =$ SYNC2) **and** ($|Halt_i| \le t$) **and** (STATE = SYNC2 for every message in $msgSet_i$) **then**
15: decide(est_i); send(k_i+1, est_i, DECIDE, $\emptyset$) to $\Pi \backslash p_i$; **return** $\{decision\}$
16: **if** $|Halt_i| \le k_i - 1$ **then**
17: $\text{STATE}_i \leftarrow$ SYNC2
18: **if** $k_i \le |Halt_i| \le t$ **then**
19: $\text{STATE}_i \leftarrow$ SYNC1
20: **if** $|Halt_i| > t$ **then**
21: $\text{STATE}_i \leftarrow$ NSYNC
22: **if** (STATE = NSYNC) **and** (received any (k_i, est', SYNC2, $*$)) **then**
23: $est_i \leftarrow est'$
24: $\text{propose}_C(est_i)$; **return** $\{decision\}$

25: **Task 2**
26: **upon** receiving (k', est', DECIDE, $*$) and ($k_i \ge k'+1$) **do**
27: stop $\text{propose}_C(*)$; decide(est'); send(k_i, est', DECIDE, $\emptyset$) to $\Pi \backslash p_i$; **return** $\{decision\}$

Fig. 1. A Consensus algorithm A_{f+2} in *ES*

In the first $t+2$ rounds, the processes exchange these three variables and then update their variables depending on the messages received. At the end of round $t+2$, if a process p_i has not yet decided, then p_i invokes the underlying uniform consensus algorithm C with the latest value of est_i as the proposal value. From the termination property of C, we know that if p_i does not crash then it eventually decided in C. The algorithm ensures the following *elimination property*: if a process completes some round $k < t+2$ with STATE = SYNC2 and $est = est'$ and no process decides in round k or in a lower round, then every process which completes round k with STATE = SYNC1 has $est \ge est'$, and every process which completes round k with STATE = SYNC2 has $est = est'$. (Processes which complete round k with STATE = NSYNC might have $est < est'$.)

We give a proof of the elimination property and that of the correctness of A_{f+2} in the full-version of the paper [6]. We now briefly discuss how the uniform agreement property of A_{f+2} follows from the elimination property. If every process which decides, decides at a round higher than $t+2$ then uniform agreement follows from the corresponding property of algorithm C. Consider the lowest round $k' \le t+2$ in which some process p_i decides, say d. From line 14, at least $n-t$ processes (a majority) send round k' messages with STATE = SYNC2, and hence, every process which completes round k' receives at least one round k' message with STATE $=$ SYNC2. From line 11, every process with STATE $=$ SYNC1 or STATE $=$ SYNC2, only considers messages with STATE $=$ SYNC1 or STATE $=$ SYNC2 while updating its est (i.e., ignores messages with STATE $=$ NSYNC). From the elimination property we know that, every round k' message with STATE $=$ SYNC1 has $est \geq d$, and every round k' message with STATE $=$ SYNC2 has $est = d$. Since every process receives a message with STATE $=$ SYNC2, processes with STATE $=$ SYNC1 or STATE $=$ SYNC2 adopts d as their est. Furthermore, every process which completes round k' with STATE $=$ NSYNC receives at least one message with STATE $=$ SYNC2 and $est = d$ and updates its est to d (line 22). Consequently, every process which completes round k', does so with $est = d$, and no value distinct from d can be decided at round k' or at a higher round.

References

1. Aguilera M. K. and Toueg S., A Simple Bivalency Proof that t-Resilient Consensus Requires $t+1$ Rounds. *Information Processing Letters (IPL)*, 71(3-4):155-158, 1999.
2. Chandra T. D. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM (JACM)*, 43(2):225-267, 1996.
3. Charron-Bost B. and Schiper A., Uniform Consensus Harder than Consensus. *Technical Report ID: DSC/2000/028*, Swiss Federal Institute of Technology in Lausanne (EPFL), 2000.
4. Dolev D., Reischuk R. and Strong R., Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720-741, 1990.
5. Dutta P. and Guerraoui R., The Inherent Price of Indulgence. *Proc. 21st ACM Symposium on Principles of Distributed Computing, (PODC'02)*, ACM Press, pp. 88-97, Monterey (CA), 2002.
6. Dutta P., Guerraoui R. and Pochon B., Early Local Decisions in Distributed Agreement. *Technical Report ID: 200324*, School of Computer and Communication Sciences, Swiss Federal Institute of Technology in Lausanne (EPFL). Available at: `http://ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200324.pdf`.
7. Dwork C., Lynch N. and Stockmeyer L., Consensus in the Presence of Partial Synchrony. *Journal of the ACM (JACM)*, 35(2):288-323, 1988.
8. Fischer M., Lynch N., A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters (IPL)*, 14(4):183-186, 1982.
9. Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374-382, 1985.
10. Hadzilacos V., Byzantine Agreement Under Restricted Types of Failures (Not Telling the Truth is Different from Telling Lies.) *Technical Report ID: TR-19-83*, Aiken Computation Laboratory, Harvard University.
11. Hadzilacos V., On the relationship between the Atomic Commitment and Consensus problems. *Proc. 9th International Workshop on Fault-Tolerant Computing*, Springer Verlag (LNCS 448), pp. 201-208, 1987.
12. Keidar I. and Rajsbaum S., Open Questions on Consensus Performance in Well-Behaved Runs. *Future Directions in Distributed Computing (FuDiCo)*, Springer Verlag (LNCS 2584).

13. Keidar I. and Rajsbaum S., A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters (IPL)*, 85(1):47-52, 2003.
14. Lynch N., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Francisco (CA), 872 pages, 1996.
15. Lamport L., The Part-Time Parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133-169, 1998.
16. Lamport L., Shostak R. and Pease M., The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382-401, 1982.
17. Moses Y. and Rajsbaum S., A Layered Analysis of Consensus. *SIAM Journal on Computing*, 31(4):989-1021, 2002.