Strongly Terminating Early-Stopping ıt k-Set Agreement in Synchronous Systems with General Omission Failures

Philippe Raipin Parvédy, Michel Raynal, Corentin Travers

Editorial Manager(tm) for Theory of Computing Systems
Manuscript Draft

Manuscript Number:

Title: Strongly Terminating Early-Stopping k-set Agreement in Synchronous Systems with General Omission Failures

Article Type: Original Research

Keywords: Agreement problem; Crash failure; Strong Termination;
Early decision; Early stopping; Efficiency; k-set agreement;
Message-passing system; Receive omission failure; Round-based computation; Send omission failure;
Synchronous system.

Corresponding Author: Michel Raynal,

Corresponding Author's Institution: Université de Rennes 1

First Author: Michel Raynal, Professor

Order of Authors: Michel Raynal, Professor; Michel Raynal; Rapin Parvedy Philippe, PhD

Strongly Terminating Early-Stopping k-set Agreement in Synchronous Systems with General Omission Failures*

Philippe Raïpin Parvédy Michel Raynal Corentin Travers

IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France
{praipinp|raynal|ctravers}@irisa.fr

Abstract

The k-set agreement problem is a generalization of the consensus problem: considering a system made up of n processes where each process proposes a value, each non-faulty process has to decide a value such that a decided value is a proposed value, and no more than k different values are decided. It has recently be shown that, in the crash failure model, $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ is a lower bound on the number of rounds for the non-faulty processes to decide (where t is an upper bound on the number of process crashes, and f, $0 \leq f \leq t$, the actual number of crashes).

This paper considers the k-set agreement problem in synchronous systems where up to $t < n/2$ processes can experience general omission failures (i.e., a process can crash or omit sending or receiving messages). It first introduces a new property, called *strong termination*. This property is on the processes that decide. It is satisfied if, not only every non-faulty process, but any process that neither crashes nor commits receive omission failure decides.The paper then presents a k-set agreement protocol that enjoys the following features. First, it is strongly terminating (to our knowledge, it is the first agreement protocol to satisfy this property, whatever the failure model considered). Then, it is *early deciding and stopping* in the sense that a process that either is non-faulty or commits only send omission failures decides and halts by round $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$. To our knowledge, this is the first early deciding k-set agreement protocol for the general omission failure model. Moreover, the protocol provides also the following additional *early stopping* property: a process that commits receive omission failures (and does not crash) executes at most $\min(\lceil \frac{f}{k} \rceil + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds. It is worth noticing that the protocol allows each property (strong termination *vs* early deciding/stopping *vs* early stopping) not to be obtained at the detriment of the two others.

The combination of the fact that $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ is lower bound on the number of rounds in the crash failure model, and the very existence of the proposed protocol has two noteworthy consequences. First, it shows that, although the general omission failure model is more severe than the crash failure model, both models have the same lower bound for the non-faulty processes to decide. Second, it shows that, in the general omission failure model, that bound applies also the processes that do not crash and commit only send omission failures.

Keywords: Agreement problem, Crash failure, Strong Termination, Early decision, Early stopping, Efficiency, k-set agreement, Message-passing system, Receive omission failure, Round-based computation, Send omission failure, Synchronous system.

*An extended abstract of a first version (15 pages) of this paper has appeared in the proceedings of SIROCCO 2006 [29].

1 Introduction

Context of the paper *k-set and consensus problems.* The k-set agreement problem generalizes the uniform consensus problem (that corresponds to the case $k = 1$). It has been introduced by S. Chaudhuri who, considering the crash failure model, investigated how the number of choices (k) allowed to the processes is related to the maximum number (t) of processes that can be faulty (i.e., that can crash) [7]. The problem can be defined as follows. Each of the n processes (processors) defining the system starts with its own value (called "proposed value"). Each process that does not crash has to decide a value (termination), in such a way that a decided value is a proposed value (validity) and no more than k different values are decided (agreement)[1].

k-set agreement can trivially be solved in crash-prone asynchronous systems when $k > t$ [7]. A one communication step protocol is as follows: (1) $t + 1$ processes are arbitrarily selected prior to the execution; (2) each of these processes sends its value to all processes; (3) a process decides the first value it receives. Differently, it has been shown that there is no solution in these systems as soon as $k \leq t$ [5, 17, 32]. (The asynchronous consensus impossibility, case $k = 1$, was demonstrated before, using a different technique [11]. A combinatorial characterization of the tasks which are solvable in presence of one process crash is presented in [3]). Several approaches have been proposed to circumvent the impossibility to solve the k-set agreement problem in process crash prone asynchronous systems (e.g., probabilistic protocols [22], or unreliable failure detectors with limited scope accuracy [16, 21, 33]).

The situation is different in process crash prone synchronous systems where the k-set agreement problem can always be solved, whatever the value of t with respect to k. It has also been shown that, in the worst case, the lower bound on the number of rounds (time complexity measured in communication steps) is $\lfloor t/k \rfloor + 1$ [8]. (This bound generalizes the $t + 1$ lower bound associated with the consensus problem [1, 2, 10, 20]. See also [4] for the case $t = 1$.)

Early decision. Although failures do occur, they are rare in practice. For the uniform consensus problem ($k = 1$), this observation has motivated the design of early deciding synchronous protocols [6, 9, 19, 30], i.e., protocols that can cope with up to t process crashes, but decide in less than $t+1$ rounds in favorable circumstances (i.e., when there are few failures). More precisely, these protocols allow the processes to decide in $\min(f + 2, t + 1)$ rounds, where f is the number of processes that crash during a run, $0 \leq f \leq t$, which has been shown to be optimal (the worst scenario being when there is exactly one crash per round) [6, 18][2].

In a very interesting way, it has very recently been shown that the early deciding lower bound for the k-set agreement problem in the synchronous crash failure model is $\lfloor f/k \rfloor + 2$ for $0 \leq \lfloor f/k \rfloor \leq \lfloor t/k \rfloor - 2$, and $\lfloor f/k \rfloor + 1$ otherwise [12]. This lower bound, not only generalizes the corresponding uniform consensus lower bound, but also shows an "inescapable tradeoff" among the number t of crashes tolerated, the number f of actual crashes, the degree k of coordination we want to achieve, and the best running time achievable [8]. As far as the time/coordination degree tradeoff is concerned, it is important to notice that, when compared to consensus, k-set agreement divides the running time by k (e.g., allowing two values to be decided halves the running time).

[1] A process that decides and thereafter crashes is not allowed to decide one more value, in addition to the k allowed values. This is why k-set agreement generalizes *uniform consensus* where no two processes (be they faulty or not) can decide different values. Non-uniform consensus allows a faulty process to decide a value different from the value decided by the correct processes. The non-uniform version of the k-set agreement problem has not been investigated in the literature.

[2] More precisely, the lower bound is $f + 2$ when $f \leq t - 2$, and $f + 1$ when $f = t - 1$ or $f = t$.

Related work While not-early deciding k-set agreement protocols for the synchronous crash failure model (i.e., protocols that always terminate in $\lfloor t/k \rfloor + 1$ rounds) are now well understood [2, 8, 20], to our knowledge, so far only two early deciding k-set agreement protocols have been proposed [13, 27] for that model. The protocol described in [13] assumes $t < n-k$, which means that the number of crashes t that can be tolerated decreases as the coordination degree k increases. The protocol described in [27], which imposes no constraint on t (i.e., $t < n$), is based on a mechanism that allows the processes to take into account the actual pattern of crash failures and not only their number, thereby allowing the processes to decide in much less than $\lfloor f/k \rfloor + 2$ rounds in a lot of cases (the worst case being only when the crashes are evenly distributed in the rounds with k crashes per round). We have recently designed an early deciding k-set agreement protocol for the synchronous send (only) omission failure model [28]. A survey of the k-set agreement problem in synchronous systems prone to crash, send omission or general omission failures is presented in [31].

Content of the paper This paper investigates the k-set agreement problem in synchronous systems prone to general omission failures and presents a k-set protocol suited to this model. This failure model lies between the crash failure model and the Byzantine failure model [24]: a faulty process is a process that crashes, or omits sending or receiving messages [14, 25]. This failure model is particularly interesting as it provides the system designers with a realistic way to represent overflow failures of the output buffers (send omission failures) or input buffers (receive omission failures) of at most t processes [14, 25]. The proposed protocol enjoys several noteworthy properties.

- The usual termination property used to define an agreement problem concerns only the correct processes: they all have to decide. This requirement is tied to the problem, independently of a particular model. Due to the very nature of the corresponding faults, there is no way to force a faulty process to decide in the crash failure model. It is the same in the Byzantine failure model where a faulty process that does not crash can decide an arbitrary value.

  The situation is different in the general omission failure model where a faulty process that does not crash cannot have an arbitrary behavior. On one side, due to the very nature of the receive omission failures committed by a process, there are runs where that process can forever be prevented from learning that it can decide a value without violating the agreement property (at most k different values are decided)[3]. So, for such a process, the best that can be done in the general case is either to decide a (correct) value, or halt without deciding because it does not know whether it has a value that can be decided. On the other side, a process that commits only send omission failures receives all the messages sent to it, and should consequently be able to always decide a correct value.

  We say that a protocol is *strongly terminating* if it forces to decide all the processes that neither crash nor commit receive omission failures(we call them the *good* processes; the other processes are called *bad* processes). This new termination criterion is both theoretically and practically relevant: it extends the termination property to all the processes that are committing only "benign" faults. The proposed protocol is strongly terminating[4].

- Although, as discussed before, early decision be an interesting property, some early-deciding (consensus) protocols make a difference between early decision and early stopping: they allow

[3] A process that commits receive omission failures, has an "autism" behavior. If it receives no message, it is isolated from the other processes and cannot learn values from its environment.

[4] None of the uniform consensus protocols for the synchronous general omission failure model that we are aware of (e.g., [25, 26]) is strongly terminating.

a correct process to decide in $\min(f+2, t+1)$ but stop only at a later round (e.g., [9]). Here we are interested in early-deciding protocols in which a process decides and stops during the very same round. More precisely, the proposed protocol has the following property:

- A good process decides and halts by round $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$.

So, when $\lfloor \frac{f}{k} \rfloor \leq \lfloor \frac{t}{k} \rfloor - 2$, the protocol has the noteworthy property to extend the $\lfloor \frac{f}{k} \rfloor + 2$ lower bound for a correct process to decide (1) from the crash failure model to the general omission failure model, and (2) from the correct processes to all the good processes.

As noticed before, it is not possible to force a bad process to decide. So, for these processes the protocol "does its best", namely it ensures the following early stopping property:

- No process executes more than $\min(\lceil \frac{f}{k} \rceil + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds.

Let us notice that it is possible that a bad process decides just before halting. Moreover, when $f = x\ k$ where x is an integer (which is always the case for consensus), or when there is no fault ($f = 0$), a bad process executes no more rounds than a good process. In the other cases, it executes at most one additional round.

- Each message carries a proposed value and two boolean arrays of size n (sets of process identities). This means that, if we do not consider the size of the proposed values (that does not depends on the protocol), the bit complexity is upper bounded by $O(n^2 f/k)$ per process.

The design of a protocol that satisfies, simultaneously and despite process crashes and general omission faults, the agreement property of the k-set problem, strong termination, early decision and stopping for the good processes and early stopping for the bad processes is not entirely obvious, as these properties are partly antagonistic. This is due to the fact that agreement requires that no more than k distinct values be decided (be the deciding processes correct or not), strong termination requires that, in addition to the correct processes, a well defined class of faulty processes decide, and early stopping requires the processes to halt as soon as possible. Consequently the protocol should not prevent processes from deciding at different rounds, and so, after it has decided, a process can appear to the other processes as committing omission failures, while it is actually correct. Finally, the strong termination property prevents the elimination from the protocol of a faulty process that commits only send omission failures as soon as it has been discovered faulty, as that process has to decide a value if it does not crash later. A major difficulty in the design of the protocol consists in obtaining simultaneously all these properties and not each one at the price of not satisfying one of the others.

General transformations from a synchronous failure model to another synchronous failure model (e.g., from omission to crash) are presented in [23]. These transformations are general (they are not associated with particular problems) and have a cost (simulating a round in the crash failure model requires two rounds in the more severe omission failure model). So, they are not relevant for our purpose.

When instantiated with $k = 1$, the protocol provides a new uniform consensus protocol for the synchronous general omission failure model. To our knowledge, this is the first uniform consensus protocol that enjoys strong termination and directs all the processes to terminate by round $\min(f+2, t+1)$. Let us finally observe that the paper leaves open two problems for future research. The first consists in proving or disproving that $\lceil \frac{f}{k} \rceil + 2$ is a tight lower bound for a bad process to stop when $f = k\ x + y$ with x and y being integers and $0 < y < k$ (we think it is). The second problem concerns t: is $t < n/2$ a lower bound to solve the strongly terminating early stopping k-set problem? (Let us remark that the answer is "yes" for $k = 1$ [23, 30].)

A k-set protocol can be useful to allocate shared resources. As an example, let us consider the allocation of broadcast frequencies in communication networks (this example is taken from [20]). Such a protocol allows processes to agree on a small number of frequencies for broadcasting large data (e.g., a movie). As the communication is broadcast-based, the processes can receive the data using the same frequency.

Roadmap The paper consists of 6 sections. Section 2 presents the computation model and gives a definition of the k-set agreement problem. To underline its basic design principles and make its understanding easier, the protocol is presented incrementally. Section 3 presents first a strongly terminating k-set agreement protocol. Then, Section 5 enriches this basic protocol to obtain an strongly terminating, early stopping k-set agreement protocol. Formal statements of the properties (lemmas and theorems) of both protocols are provided in Section 4 and Section 6, respectively. The proofs of the protocols are also done incrementally.

2 Computation Model and Strongly Terminating k-Set Agreement

2.1 Round-Based Synchronous System

The system model consists of a finite set of processes, namely, $\Pi = \{p_1, \ldots, p_n\}$, that communicate and synchronize by sending and receiving messages through channels. Every pair of processes p_i and p_j is connected by a channel denoted (p_i, p_j). The underlying communication system is assumed to be failure-free: there is no creation, alteration, loss or duplication of message.

The system is *synchronous*. This means that each of its executions 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 messages.
- 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 same round r.
- A computation phase during which each process processes the messages it received during that round and executes local computation.

2.2 Process Failure Model

A process is *faulty* during an execution if its behavior deviates from that prescribed by its algorithm, otherwise it is *correct*. A *failure model* defines how a faulty process can deviate from its algorithm [15]. We consider here the following failure models:

- Crash failure. A faulty process stops its execution prematurely. After it has crashed, a process does nothing. Let us observe that 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 sent.
- Send Omission failure. A faulty process crashes or omits sending messages it was supposed to send [14].

- General Omission failure. A faulty process crashes, omits sending messages it was supposed to send or omits receiving messages it was supposed to receive (receive omission) [25].

It is easy to see that these failure models are of increasing "severity" in the sense that any protocol that solves a problem in the General Omission (resp., Send Omission) failure model, also solves it in the (less severe) Send Omission (resp., Crash) failure model [15]. This paper considers the General Omission failure model. As already indicated, n, t and f denote the total number of processes, the maximum number of processes that can be faulty, and the actual number of processes that are faulty in a given run, respectively ($0 \leq f \leq t < n/2$).

As defined in the introduction, a *good* process is a process that neither crashes nor commits receive omission failures. A *bad* process is a process that commits receive omission failures or crashes. So, given a run, each process is either good or bad. A good process commits only "benign" failures, while a bad process commits "severe" failures.

2.3 Strongly Terminating k-Set Agreement

The problem has been informally stated in the Introduction: every process p_i *proposes* a value v_i and each correct process has to *decide* on a value in relation to the set of proposed values. More precisely, the k-set agreement problem is defined by the following three properties:

- Termination: Every correct process decides.
- Validity: If a process decides v, then v was proposed by some process.
- Agreement: No more than k different values are decided.

As we have seen 1-set agreement is the uniform consensus problem. In the following, we implicitly assume $k \leq t$ (this is because, as we have seen in the introduction, k-set agreement is trivial when $k > t$).

As already mentioned, we are interested here in protocols that direct all the good processes to decide. So, we consider a stronger version of the k-set agreement problem, in which the termination property is replaced by the following property:

- Strong Termination: Every good process decides.

3 A Strongly Terminating k-Set Agreement Protocol

We first present a strongly terminating k-set agreement protocol where the good processes terminate in $\lfloor \frac{t}{k} \rfloor + 1$ rounds. The protocol is described in Figure 1. r is a global variable that defines the current round number; the processes can only read it.

A process p_i starts the protocol by invoking the function k-SET_AGREEMENT(v_i) where v_i is the value it proposes. It terminates either by crashing, by returning the default value $\perp$ at line 08, or by returning a proposed value at line 11. As we will see, only a bad process can exit at line 08 and return $\perp$. That default value cannot be proposed by a process. So, returning $\perp$ means "no decision" from the k-set agreement point of view.

3.1 Local Variables

A process p_i manages four local variables. The scope of the first two is the whole execution, while the scope of the last two is limited to each round. Their meaning is the following:

- est_i is p_i's current estimate of the decision value. It is initial value is v_i (line 01).
- $trusted_i$ represents the set of processes that p_i currently considers as being correct. It initial value is Π (the whole set of processes). So, $i \in trusted_i$ (line 04) means that p_i considers it is correct. If $j \in trusted_i$ we say "p_i trusts p_j"; if $j \notin trusted_i$ we say "p_i suspects p_j".
- rec_from_i is a round local variable used to contain the ids of the processes that p_i does not currently suspect and from which it has received messages during that round (line 05).
- $W_i(j)$ is a set of processes identities that represents the set of the processes p_ℓ that are currently trusted by p_i and that (to p_i's knowledge) trust p_j (line 06).

3.2 Process Behavior

The aim is for a process to decide the smallest value it has seen. But, due to the send and receive omission failures possibly committed by some processes, a process cannot safely decide the smallest value it has ever seen, it can only safely decide the smallest in a subset of the values it has received during the rounds. The crucial part of the protocol consists in providing each process with correct rules that allow it to determine its "safe subset".

```
Function k-SET_AGREEMENT(v_i)
(01) est_i ← v_i; trusted_i ← Π;    % r = 0 %
(02) for r = 1, ..., ⌊t/k⌋ + 1 do
(03) begin_round
(04)    if (i ∈ trusted_i) then foreach j ∈ Π do send(est_i, trusted_i) to p_j enddo endif;
(05)    let rec_from_i = {j : (est_j, trust_j) is received from p_j during r ∧ j ∈ trusted_i};
(06)    foreach j ∈ rec_from_i let W_i(j) = {ℓ : ℓ ∈ rec_from_i  ∧  j ∈ trust_ℓ};
(07)    trusted_i ← rec_from_i − {j : |W_i(j)| < n − t};
(08)    if (|trusted_i| < n − t) then return (⊥) endif;
(09)    est_i ← min(est_j received during r and such that j ∈ trusted_i)
(10) end_round;
(11) return (est_i)
```

Figure 1: Strongly terminating k-set protocol for general omission failures, code for p_i, $t < \frac{n}{2}$

During each round r, these rules are implemented by the following process behavior decomposed in three parts according to the synchronous round-based computation model.

- If p_i considers it is correct ($i \in trusted_i$), it first sends to all the processes its current local state, namely, the current pair $(est_i, trusted_i)$ (line 04). Otherwise, p_i skips the sending phase.
- Then, p_i executes the receive phase (line 05). As already indicated, when it considers the messages it has received during the current round, p_i considers only the messages sent by the the processes it trusts (here, the set $trusted_i$ can be seen as a filter).
- Finally, p_i executes the local computation phase that is the core of the protocol (lines 06-09). This phase is made up of the following statements where the value $n-t$ constitutes a threshold that plays a fundamental role.
  - First, p_i determines the new value of $trusted_i$ (lines 06-07). It is equal to the current set rec_from_i from which are suppressed all the processes p_j such that $|W_i(j)| < n - t$. These processes p_j are no longer trusted by p_i because there are "not enough" processes

trusted by p_i that trust them (p_j is missing "Witnesses" to remain trusted by p_i, hence the name $W_i(j)$); "not enough" means here less than $n - t$.

- Then, p_i checks if it trusts enough processes, i.e., at least $n - t$ (line 08). If the answer is negative, as we will see in the proof, p_i knows that it has committed receive omission failures and cannot safely decide. It consequently halts, returning the default value $\perp$.
- Finally, if it has not stopped at line 08, p_i computes its new estimate of the decision value (line 09) according to the estimate values it has received from the processes it currently trusts.

4 Proof of the Strongly Terminating Protocol

The protocol proof assumes $t < n/2$. It uses the following notations.

- Given a set of process identities $X = \{i, j, \dots\}$, we sometimes use $p_i \in X$ for $i \in X$.
- $\mathcal{C}$ is the set of correct processes in a given execution.
- $x_i[r]$ denotes the value of p_i's local variable x at the end of round r.
  By definition $trusted_i[0] = \Pi$. When $j \in trusted_i$, we say that "p_i trusts p_j" (or "p_j is trusted by p_i").
- $Completing[r] = \{i : p_i \text{ proceeds to } r+1 \}$. By definition $Completing[0] = \Pi$. (If $r = \lfloor \frac{t}{k} \rfloor + 1$, "$p_i$ proceeds to $r + 1$" means p_i executes line 11.)
- $EST[r] = \{est_i[r] : \ i \in Completing[r]\}$. By definition $EST[0] =$ the proposed values.
  $EST[r]$ contains the values that are present in the system at the end of round r.
- $Silent[r] = \{i : \ \forall j \in Completing[r] : \ i \notin trusted_j[r]\}$. It is important to remark that if $i \in Silent[r]$, then no process p_j (including p_i itself) takes into account est_i sent by p_i (if any) to update its local variables est_j at line 09 of the round r. ($Silent[0] = \emptyset$.)

The proof of the following relations are left to the reader:

$$\begin{aligned} Completing[r+1] &\subseteq Completing[r], \\ Silent[r] &\subseteq Silent[r+1], \\ \forall i \in Completing[r] : \ Silent[r] &\subseteq \Pi - trusted_i[r]. \end{aligned}$$

4.1 Basic Lemmas

The first lemma that follows will be used to prove that a process that does not commit receive omission failure decides.

Lemma 1 *Let p_i be a process that is correct or commits only send omission failures. We have $\forall r$: (1) $\mathcal{C} \subseteq trusted_i[r]$ and (2) $i \in Completing[r]$.*

Proof The proof is by induction on the round number r. Let p_i be a process that is correct or commits only send omission failures.

- Base case. Let us first observe that we have initially $\forall j : \ trusted_j[0] = \Pi$. The set $rec\_from_i[1]$ computed by p_i at line 05 of the first round includes all the processes that did not commit send omission failure: it consequently includes (at least) all the correct processes, i.e., at least $n - t$ processes.

  Let us consider any correct process p_j. That process is such that $j \in trust_\ell$, for any p_l from which p_i receives a message, because $trust_\ell$ carries the value $trusted_\ell[0]$ which is equal to Π. As there are at least $n - t$ correct processes, it follows that the set $W_i(j)$ (computed at line 06) contains at least $n - t$ processes. We can then conclude that all the correct processes p_j belong to $rec\_from_i[1]$ and none of them is suppressed from it when the value $trusted_i[1]$ is computed at line 07. It follows that $|trusted_i[1]| \geq n - t$, from which we conclude that p_i does not stop at line 08. This establishes the base case $r = 1$: for all the processes p_i that do not commit receive omission failures during the first round we have $p_i \in Completing[1]$ and $\mathcal{C} \subseteq trusted_i[1]$.

- Induction step. Let us assume that the lemma is true from the first round until round $r - 1$. We show it remains true at round r. First of all, let us notice that any correct process p_j sends a message during r. This follows from the induction assumption: as $j \in Completing[r - 1]$ and $j \in trusted_j[r - 1]$, p_j executes the broadcast at line 04 of the round r.

  The proof is then the same as the second paragraph of the base step, replacing $trusted_\ell[0]$ (equal to Π) by $trusted_\ell[r - 1]$ that contains (at least) the correct processes (induction assumption) and the pair of round numbers $(0, 1)$ by the pair $(r - 1, r)$, respectively.

$\square_{Lemma\ 1}$

The next two lemmas show that $n - t$ is a critical threshold related to the number of processes (1) for a process to become silent or (2) for the process estimates to become smaller or equal to some value. More explicitly, the first of these lemmas states that if a process p_x is not trusted by "enough" processes (i.e., trusted by less than $n - t$ processes[5]) at the end of a round $r - 1$, then that process p_x is not trusted by the processes that complete round r.

Lemma 2 $\forall r \geq 1 : \forall x : \big|\{y : y \in Completing[r-1] \wedge x \in trusted_y[r-1]\}\big| < n-t \Rightarrow x \in Silent[r]$.

Proof Given a round $r - 1$, let p_x be a process such that $\big|\{y : y \in Completing[r - 1] \wedge x \in trusted_y[r - 1])\}\big| < n - t$. Let $p_i \in Completing[r]$. We have to show that, after p_i has executed line 07, $x \notin trusted_i[r]$.

- $x \notin trusted_i[r - 1]$ or p_i does not receive a message from p_x during round r. In that case, we have $x \notin rec\_from_i[r]$. It follows from the way p_i updates its set $trusted_i$ (line 07) that $x \notin trusted_i[r]$.

- $x \in trusted_i[r - 1]$ and p_i receives a message from p_x during round r (i.e. $x \in rec\_from_i[r]$). Let us consider the set $W_i(x)$ computed by p_i at line 06 during round r. Let us observe that a process p_j that does not trust p_x at the end of round $r - 1$ sends a pair $(est_j, trusted_j)$ such that $x \notin trusted_j$. Consequently, due to the lemma assumptions, p_i receives at most t $(est, trust)$ messages such that $x \in trust$. As $t < n/2$, we have $t < n-t$, from which it follows that $|\{j : j \in rec\_from_i \wedge x \in trust_j\}| < n - t$. As $W_i(x) \subseteq \{j : j \in rec\_from_i \wedge x \in trust_j\}$ (line 06), x is removed from $trusted_i$ (line 07) and consequently $x \notin trusted_i[r]$.

[5]Equivalently, trusted by at most t processes.

$\square_{Lemma\ 2}$

The next lemma shows that if "enough" (i.e., at least $n-t$) processes have an estimate smaller than or equal to a value v at the end of a round $r-1$, then no process $p_i \in Completing[r]$ has a value greater than v at the end of r.

Lemma 3 *Let v be an arbitrary value.* $\forall r \geq 1 : \big|\{x : est_x[r-1] \leq v \ \wedge \ x \in Completing[r-1]\}\big| \geq n-t \Rightarrow \forall i \in Completing[r] : est_i[r] \leq v$.

Proof Let v be a value such that at least $n-t$ processes p_j are such that $est_j[r-1] \leq v$. Let p_j be one of these (at least $n-t$) processes that belongs to $Completing[r]$ and sends a message during r. Let us observe that the pair $(est_j, trusted_j)$ sent during r by p_j is such that $est_j \leq v$.

Let $p_i \in Completing[r]$. Due to the very definition of $Completing[r]$, p_i does not stop by returning $\perp$ at line 09, and consequently, $|trusted_i[r]| \geq n-t$. This implies that the set of est_j values received by p_i during round r and used to compute its new estimate (at line 09) contains at least $n-t$ values. Due to the "majority of correct processes" assumption ($n-t > t$) and, to the fact that two majorities always intersect, at least one of these est_j is such that $est_j \leq v$. The min() function used by p_i to update est_i at line 09 allows concluding that $est_i[r] \leq v$. $\square_{Lemma\ 3}$

Finally, the next lemma states that the sequence of set values $EST[0]$, $EST[1]$,... is monotonic and never increases.

Lemma 4 $\forall r \geq 0 : \ EST[r+1] \subseteq EST[r]$.

Proof The lemma follows directly from the fact that, during a round, values can only disappear because (1) the new value of est_i computed by a process is the smallest of values it has received, and (2) some processes may stop sending or receiving messages. $\square_{Lemma\ 4}$

4.2 Central Lemma

The lemma that follows is central to prove the agreement property, namely, at most k distinct values are decided. Its formulation is early-stopping oriented. Being general, this formulation allows using the same lemma to prove both the non-early stopping version of the protocol (Theorem 3) and the early stopping protocol (Theorem 4).

Lemma 5 *Let r ($1 \leq r \leq \lfloor \frac{t}{k} \rfloor + 1$) be a round such that (1) $\mathcal{C} \subseteq Completing[r-1]$, and (2) $|EST[r]| > k$ (let v_m denote the kth smallest value in $EST[r]$, i.e., the greatest value among the k smallest values of $EST[r]$). Let $i \in Completing[r]$. We have $n-k\ r < |trusted_i[r]| \Rightarrow est_i[r] \leq v_m$.*

Proof Let us first consider the case $r = 1$. As $p_i \in Completing[r]$ and $n-k < |trusted_i[r]|$, p_i misses at most $k-1$ values during the first round. It follows that $est_i[1] \leq v_m$.

The rest of the proof addresses the case $r \geq 2$. To prove the lemma, we prove the contrapositive, namely $est_i[r] > v_m \Rightarrow |trusted_i[r]| \leq n-k\ r$. In the following, r and p_i denote the round number and the processes introduced in the lemma statement. Let us consider the following set of processes:

$$P(v,x) = \{p_\ell : \exists x' \leq x \text{ such that } \ell \in Completing[x'] \wedge est_\ell[x'] \leq v\}$$

where v is a proposed value and x, $0 \leq x \leq \lfloor \frac{t}{k} \rfloor + 1$, a round number. ($P(v,x), x > 0$ is the set of processes that have processed a value $v' \leq v$ during some round $x' \leq x$; $P(v,0)$ is the set of processes whose initial value is smaller than or equal to v.)

Let $r \geq 1$. We claim $k\ r \leq |P(v_m, r-1)|$ (Claim *C1*) and $P(v_m, r-1) \subseteq \Pi - trusted_i[r]$ (Claim *C2*). The lemma follows directly from these claims, as combining *C1* and *C2* we obtain $k\ r \leq |P(v_m, r-1)| \leq |\Pi - trusted_i[r]|$, from which we conclude that $|trusted_i[r]| \leq n - k\ r$.

The proofs of *C1* and *C2* are based on the following properties (implicitly defined in the context of the lemma assumptions for $r \geq 2$):
Property *P1*: $\forall\ r' \leq r-2 :\ P(v_m, r') \subseteq Silent[r'+2]$,
Property *P2*: $\forall\ r' \leq r-2 :\ k \leq |P(v_m, r'+1) - P(v_m, r')|$.
We first prove *P1* and *P2*, and then prove the two claims.

Property P1: $\forall\ r' \leq r-2 :\ P(v_m, r') \subseteq Silent[r'+2]$.
Proof of P1. Let $r' \leq r-2$. We consider two cases, namely $r' < r-2$ and $r' = r-2$.

- $r' < r-2$. Let $p_x \in P(v_m, r')$. From the definition of the $P(v_m, r')$ set, there is a round $r'' \leq r'$ such that $est_x[r''] \leq v_m$. We claim that, at the end of round $r''+1$, at least $n-t$ processes do not trust p_x, which allows us to conclude from Lemma 2 that $x \in Silent[r''+2]$. The fact that $Silent[r''+2] \subseteq Silent[r'+2]$ completes the proof.

  Proof of the Claim. Let p_c be a correct process that has not decided by the end of round $r-1$. Due to the lemma assumptions, such a correct process does exist. In order to obtain a contradiction, let us suppose that p_c trusts p_x at the end of round $r''+1$ (i.e., $x \in trusted_c[r''+1]$). This implies that p_c receives and processes a message $(est_x, -)$ from p_x during round $r''+1$ and, due to the min() function used to compute a new estimate, we have $est_c[r''+1] \leq v_m$.

  Let us observe that (O1) all the correct processes have started the round r (by assumption), (O2) a correct process is trusted by every correct process (Lemma 1 and O1) and, (O3) a correct process p_y is such that $\forall d, d' : d' < d \Rightarrow est_y[d] \leq est_y[d']$ (this is because a correct process always receives and processes a message from itself).

  Let p_y a correct process. As p_y trusts p_c at round $r''+2$ (Observation O2), p_c sends an estimate $v \leq v_m$ during round $r''+2$ and, due to the min() function used to compute a new estimate, we have $est_y[r''+2] \leq v_m$. Moreover, until it decides, p_y is then such that $est_y \leq v_m$ (Observation O3). In particular, at the end of the round $r-1$, every correct process p_y is such that $est_y[r-1] \leq v_m$.

  Moreover, as there are at least $n-t$ correct processes that belong to $Completing[r-1]$ (lemma assumption), it follows from Lemma 3 that all the processes p_y that belong to $Completing[r]$ are such that $est_y[r] \leq v_m$. As p_i belongs to $Completing[r]$ (lemma assumption) we have $est_i[r] \leq v_m$: a contradiction (remind that the proof initially assumes that $est_i[r] > v_m$). Thus, at the end of round $r''+1$, for each correct process p_c, $x \notin trusted_c[r''+1]$. As at least $n-t$ correct processes belong to $Completing[r''+1]$, we conclude that $n-t$ processes do not trust p_x at the end of round $r''+1$. *End of the proof of the Claim.*

- $r' = r-2$. Let $p_x \in P(v_m, r') - P(v_m, r'-1)$ (if $p_x \in P(v_m, r'-1)$, the previous case applies). As $i \in Completing[r]$ and $est_i[r] > v_m$, taking the contrapositive of Lemma 3 we obtain $|\{y : y \in Completing[r-1] \wedge est_y[r-1] \leq v_m\}| < n-t$. It follows that, even if p_x sends $est_x[r-2] \leq v_m$ during $r-1$, strictly less than $n-t$ processes receive and process that message from p_x during $r-1$. This implies that $|\{y : x \in trusted_y[r-1]\}| < n-t$, from which we conclude by applying Lemma 2, that $x \in Silent[r]$.

End of the proof of the property P1.

Property P2: $\forall\ 0 \leq r' \leq r-2 : k \leq |P(v_m, r'+1) - P(v_m, r')|$.
Proof of P2. Let r' be a round number, $0 \leq r' \leq r-2$ and $p_x \in P(v_m, r')$. From property $P1$, we know that $p_x \in Silent[r'+2]$. Thus, during $r'+2$, any process $p_j \in Completing[r'+2]$ (possibly including p_x itself) ignore the round $r'+1$ estimate of p_x (i.e., $est_x[r'+1]$) to compute $est_j[r'+2]$. It follows that, if all the processes p_j such that $est_j[r'+1] \leq v_m$ were such that $p_j \in P(v_m, r')$, then no value $v \leq v_m$ would belong to $EST[r'+2]$. This means that the only possibility for such values to belong to $EST[r'+2]$, is to be adopted during $r'+1$ by some $p_y \notin P(v_m, r')$.

As $EST[r] \subseteq EST[r'+2]$ (Lemma 4), and $EST[r]$ contains k values smaller than or equal to v_m (lemma assumption), we can conclude that $EST[r'+2]$ contains at least k values smaller than or equal to v_m. It follows that, during round $r'+1$, at least k processes p_j such that $p_j \in Completing[r'+1] \wedge p_j \notin P(v_m, r')$ adopt an estimate smaller than or equal to v_m. This implies that $|P(v_m, r'+1) - P(v_m, r')| \geq k$. *End of the proof of the property P2.*

Claim C1: $k\ r \leq |P(v_m, r-1)|$.
Proof of C1. The proof is by induction on the round number r'.

- Base case $r' = 1$: By assumption, there are k distinct values smaller than or equal to v_m in $EST[r]$. As no new value appears in a round, at least k distinct values smaller than or equal to v_m were initially proposed, it follows that $k \leq |P(v_m, 0)|$.
- Induction case: $k\ r' \leq |P(v_m, r'-1)|$ is satisfied for $1 \leq r' < r$.
  As $k \leq |P(v_m, r') - P(v_m, r'-1)|$ (Property $P2$) and as $P(v_m, r'-1) \subseteq P(v_m, r')$ (from the definition of the $P(v, x)$ sets), we have $k + |P(v_m, r'-1)| \leq |P(v_m, r')|$ (A).

  Combining $k(r-1) \leq |P(v_m, r-2)|$ (induction assumption) with A, we obtain $k\ r \leq |P(v_m, r-1)|$.

End of the proof of the claim C1.

Claim C2: $P(v_m, r-1) \subseteq \Pi - trusted_i[r]$.
Proof of C2. The claim is trivially satisfied if $trusted_i[r] = \emptyset$. In the other case, let us observe that, as $p_i \in Completing[r]$, we have $Silent[r] \subseteq \Pi - trusted_i[r]$ (see the "Notation" section). Combining this inclusion with $P(v_m, r-2) \subseteq Silent[r]$ (Property $P1$), we obtain $P(v_m, r-2) \subseteq \Pi - sender_i[r]$ (B).

Due to the property $P2$, the set $P(v_m, r-1) - P(v_m, r-2)$ has at least k elements. Hence, it is not empty. Let $p_x \in P(v_m, r-1) - P(v_m, r-2)$. We consider two cases. If p_x does not send a message to p_i or p_i fails to receive the message of p_x during r, we have $x \notin rec\_from_i[r]$ which implies $x \in \Pi - trusted_i[r]$ (line 07). If p_x sends a message to p_i in round r, it sends $v = est_x[r-1] \leq v_m$. Due to the min() function used to compute new estimate (line 09) and as p_i is such that $est_i[r] > v_m$, p_i does not process v during r. It follows that $x \in \Pi - trusted_i[r]$. So, for each process p_x such that $p_x \in P(v_m, r-1) - P(v_m, r-2)$, we always have $x \in \Pi - trusted_i[r]$ (C). Combining B and C, we obtain $P(v_m, r-1) \subseteq \Pi - trusted_i[r]$ which proves the claim. *End of the proof of the claim C2.*

$\square_{Lemma\ 5}$

4.3 Properties of the Protocol

Theorem 1 [Validity] *A decided value is a proposed value.*

Proof Let us first observe that a process p_i decides at line 11 of the last round. It then decides $est_i[\lfloor \frac{t}{k} \rfloor + 1]$.

The proof is an easy induction on the round number. Initially ($r = 0$), each est_i local variable contains a proposed value (line 01). Let us assume this is true until round $r - 1$. We show it is true at the end of round r. Let us notice that, due to the test of line 08, p_i updates est_i at line 09 only if $|trusted_i| \geq n - t$ (otherwise, p_i stops at line 08 without deciding). Due to line 07, $trusted_i$ is a set including only processes p_j whose value est_j has been received during the current round r. As that value is the value computed by p_j during the previous round, it follows from the induction assumption that est_i contains a proposed value. $\square_{Theorem\ 1}$

Theorem 2 [Strong Termination] *A process p_i that neither crashes nor commits receive omission failures decides.*

Proof Let p_i be a good process (so, either p_i is correct, or commits only send omission failures). Lemma 1 shows that $\forall r : \mathcal{C} \subseteq trusted_i[r]$. We conclude from that lemma that $\forall r : |trusted_i[r]| \geq |\mathcal{C}| \geq n - t$. It follows that p_i never exits at line 08. Consequently, p_i decides at line 11 of the last round $r = \lfloor \frac{t}{k} \rfloor + 1$. $\square_{Theorem\ 2}$

As a correct process does not commit receive omission failures, the following corollary is an immediate consequence of the previous theorem.

Corollary 1 [Termination] *Every correct process decides.*

Theorem 3 [Agreement] *No more than k different values are decided.*

Proof Let us consider the set $EST[\lfloor \frac{t}{k} \rfloor + 1]$ that contains the estimate values present in the system at the end of the round $\lfloor \frac{t}{k} \rfloor + 1$. We claim $|EST[\lfloor \frac{t}{k} \rfloor + 1]| \leq k$ (claim C). Due to very definition of the $EST[r]$ sets, a process that decides decides a value $\in EST[\lfloor \frac{t}{k} \rfloor + 1]$. This implies that at most k different values are decided.

Proof of C. Let $t = kx+y$ with $y < k$ (hence $\lfloor \frac{t}{k} \rfloor = x$). The proof is by contradiction. Let us assume that $|EST[x+1]| > k$. Let v_m be the kth smallest values in $EST[x+1]$ and let $i \in Completing[x+1]$ such that $est_i[x + 1] > v_m$.

As each correct process decides (Corollary 1), there are at least $n - t$ (correct) processes in $Completing[x + 1]$. Moreover, as $|EST[x + 1]| > k$, the assumptions of Lemma 5 are satisfied. Considering our assumption $est_i[x + 1] > v_m$, and applying the contrapositive of Lemma 5 to process p_i, we obtain $|trusted_i[x + 1]| \leq n - k(x + 1) = n - (kx + k) < n - (kx + y) = n - t$. This implies that p_i returns $\perp$ at line 08 during the round $x + 1$: a contradiction with the fact that $i \in Completing[x + 1]$. *End of the proof of the claim C.* $\square_{Theorem\ 3}$

5 A Strongly Terminating and Early Stopping k-Set Agreement Protocol

This section enriches the previous strongly terminating k-set agreement protocol to obtain an early stopping protocol, namely, a protocol where a good process decides and halts by round $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$, and a bad process executes at most $\min(\lceil \frac{f}{k} \rceil + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds.

The protocol is described in Figure 2. To make reading and understanding easier, all the lines from the first protocol appears with the same number. The line number of each of the 10 new lines that make the protocol early stopping are prefixed by "E". We explain here only the new parts of the protocol.

```
Function k-SET_AGREEMENT(v_i)
(01)   est_i ← v_i; trusted_i ← Π; can_dec_i ← ∅;   % r = 0 %
(02)   for r = 1, ..., ⌊t/k⌋ + 1 do
(03)   begin_round
(04)     if (i ∈ trusted_i) then foreach j ∈ Π do send(est_i, trusted_i, can_dec_i) to p_j enddo endif;
(E01)    let REC_FROM_i = {i} ∪ {j : (est_j, trust_j, c_dec_j) is received from p_j during r};
(E02)    let CAN_DEC_i = ∪(c_dec_j : j ∈ REC_FROM_i);
(E03)    if (i ∉ trusted_i ∨ i ∈ can_dec_i) then
(E04)          if |CAN_DEC_i| > t then let EST_i = {est_j : j ∈ REC_FROM_i ∧ c_dec_j ≠ ∅};
(E05)                                  return (min(EST_i))
(E06)    endif endif ;
(05)     let rec_from_i = {j : (est_j, trust_j, c_dec_j) is received from p_j during r ∧ j ∈ trusted_i};
(06)     foreach j ∈ rec_from_i let W_i(j) = {ℓ : ℓ ∈ rec_from_i ∧ j ∈ trust_ℓ};
(07)     trusted_i ← rec_from_i − {j : |W_i(j)| < n − t};
(08)     if (|trusted_i| < n − t) then return (⊥) endif;
(09)     est_i ← min(est_j received during r and such that j ∈ trusted_i);
(E07)    can_dec_i ← ∪(c_dec_j received during r and such that j ∈ trusted_i);
(E08)    if (i ∈ trusted_i ∧ i ∉ can_dec_i)
(E09)      then if (n − k r < |trusted_i|) ∨ (can_dec_i ≠ ∅) then can_dec_i ← can_dec_i ∪ {i} endif
(E10)    endif
(10)   end_round;
(11)   return (est_i)
```

Figure 2: k-set early-deciding protocol for general omission failures, code for p_i, $t < \frac{n}{2}$

5.1 Additional Local Variables

A process p_i manages three additional local variables, one (can_dec_i) whose scope is the whole computation, and two (CAN_DEC_i and REC_FROM_i) whose scope is limited to each round. Their meaning is the following.

- can_dec_i is a set of process identities that contains, to p_i's knowledge, all the processes that can decide a value without violating the agreement property. The current value of can_dec_i is part of each message sent by p_i. Its initial value is $\emptyset$.
- REC_FROM_i is used by p_i to store its id plus the ids of all the processes from which it has received messages during the current round r (line E01). Differently from the way rec_from_i is computed (line 05), no filtering (with the set $trusted_i$) is used to compute REC_FROM_i.
- CAN_DEC_i is used to store the union of all the can_dec_j sets that p_i has received during the current round r (line E02).

5.2 Process Behavior

As already indicated, the behavior of a process p_i is modified by adding only 10 lines (E01-E10). It is important to notice that no variable used in the basic protocol is updated by these lines; the basic protocol variables are only read. This means that, when there is no early deciding/stopping at line E05, the enriched protocol behaves exactly as the basic protocol.

Let us now examine the two parts of the protocol where the new statements appear.

- Let us first consider the lines E07-E10.
  After it has updated its current estimate est_i (line 09), p_i updates similarly its set can_dec_i, to learn the processes that can early decide. As we can see, est_i and can_dec_i constitute a pair that is sent (line 04) and updated "atomically".

  Then, if p_i trusts itself ($i \in trusted_i$) and, up to now, was not allowed to early decide and stop ($i \notin can\_dec_i$), it tests a predicate to know if it can early decide. If it can, p_i adds its identity to can_dec_i (line E09). The "early decision" predicate is made up of two parts:

  - If $can\_dec_i \neq \emptyset$, then p_i learns that other processes can early decide. Consequently, as it has received and processed their estimates values (line 09), it can safely adds its identity to can_dec_i.
  - If $n - k\ r < |trusted_i|$, then p_i discovers that the set of processes it trusts is "big enough" for it to conclude that it knows one of the k smallest estimate values currently present in the system. "Big enough" means here greater than $n - k\ r$. (Let us notice that that threshold was used in Lemma 5 in the proof of the basic protocol.)

- Let us now consider the lines E01-E06.
  As already indicated REC_FROM_i and CAN_DEC_i are updated in the receive phase of the current round.

  To use these values to decide during the current round (at line E05), p_i must either be faulty (predicate $i \notin trusted_i$) or have previously sent its pair (est_i, can_dec_i) to the other processes (predicate $i \notin trusted_i \vee i \in can\_dec_i$ evaluated at line E03). But, when $i \in trusted_i$, $i \in can\_dec_i$ is not a sufficiently strong predicate for p_i to safely decide. This is because it is possible that p_i committed omission faults just during the current round. So, to allow p_i to early decide, we need to be sure that at least one correct process can decide (as it is correct such a process p_j can play a "pivot" role sending its (est_j, can_dec_j) pair to all the processes). Hence, the intuition for the final early decision/stopping predicate, namely $|CAN\_DEC_i| > t$ used at line E04: that additional predicate guarantees that at least one correct process can early decide and consequently has transmitted or will transmit its (est_j, can_dec_j) pair to all.

So, the early decision/stopping predicate for a process p_i spans actually two rounds r and r' ($r' > r$). This is a "two phase" predicate split as follows:

- During r (lines E08—E09): $(i \in trusted_i \wedge i \notin can\_dec_i) \wedge (n - k\ r < |trusted_i|) \vee (can\_dec_i \neq \emptyset)$, and
- During r' (lines E03—E04): $(i \notin trusted_i \vee i \in can\_dec_i)\ \wedge\ |CAN\_DEC_i| > t$.

Moreover, for a correct process p_i, the assignment $can\_dec_i \leftarrow can\_dec_i \cup \{i\}$ can be interpreted as a synchronization point separating the time instants when they are evaluated to *true*.

6 Proof of the Strongly Terminating Early Stopping Protocol

6.1 Basic Lemmas

The next lemma extends Lemma 1 to the early stopping context.

Lemma 6 *Let r_d be the first round during which a correct process decides at line E05 (If there is no such round, let $r_d = \lfloor \frac{t}{k} \rfloor + 1$). Let p_i be a process that is correct or commits only send omission failures. $\forall r \leq r_d$: if p_i does not decide at line E05 of the round r, we have (1) $\mathcal{C} \subseteq trusted_i[r]$ and (2) $i \in Completing[r]$.*

Proof The proof is a straightforward extension of the proof of Lemma 1. It is left to the reader. $\square_{Lemma\ 6}$

Lemma 5 considers a round r such that $\mathcal{C} \subseteq Completing[r-1]$ (i.e., a round executed by all the correct processes). Its proof relies on Lemma 1, but considers only the rounds $r' \leq r$. As, until a correct process decides, the Lemma 1 and the Lemma 6 are equivalent, it follows that the Lemma 1 can be replaced by Lemma 6 in the proof of Lemma 5. Let us also observe that the proofs of the Lemmas 2, 3 and 4 are still valid in the early stopping context (these proofs use the set $Completing[r]$ and do not rely on the set $\mathcal{C}$). We now state and prove additional lemmas used to prove the early stopping k-set agreement protocol.

Lemma 7 *The set $EST_i[r]$ computed by p_i during round r (line E04) is not empty.*

Proof Let p_i be a process and r be a round number such that p_i computes EST_i during round r. Let us first observe that, due to the test of line E04, $CAN\_DEC_i \neq \emptyset$. As, from the protocol text (line E02), $CAN\_DEC_i = \bigcup_{j \in REC\_FROM_i} c\_dec_j$, it necessarily exists x such that $x \in REC\_FROM_i \wedge c\_dec_x \neq \emptyset$. Moreover, due to the definition of $REC\_FROM_i[r]$, $x = i$ or $x \neq i$. In the first case, $est_x = est_i[r-1]$ is associated with c_dec_x. In the second case, the estimate $est_x[r-1]$ sent by p_x and received by p_i during r is associated with c_dec_x. In both case, this estimate belongs to $EST_i[r]$ (from the very definition of $EST_i[r]$ at line E04). $\square_{Lemma\ 7}$

Lemma 8 *Assuming that a process decides at line E05 during round r, let p_x be a process that proceeds to round $r+1$ (if $r = \lfloor \frac{t}{k} \rfloor + 1$, "proceed to round $r+1$" means "execute the* return() *statement at line 11"). We have: $x \notin trusted_x[r] \vee x \in can\_dec_x[r]$.*

Let us remark that it follows from that lemma that (i) if p_x executes line E03 during round $r+1$, it then evaluates the predicate in the **if** statement to true. Moreover, (ii) if p_x sends messages during round $r+1$ (which implies that $x \in trusted_x[r]$, line 04), these messages necessarily carry a can_dec_x set that contains x. **Proof** Let us define $I[r] = \{y : \exists r'_y < r$ such that $y \in can\_dec_y[r'_y]\}$, where r is the round number defined in the lemma statement. Let p_x a process that proceeds to $r+1$.

Let p_i be a process that decides during round r. As p_i decides at line E05, we have $|CAN\_DEC_i[r]| > t$. As any process p_j is the only that can start adding j in a can_dec set (line E09), it follows from the way the CAN_DEC sets are computed (lines E01-E02) that at least $t+1$ processes p_y have executed $can\_dec_y \leftarrow can\_dec_y \cup \{y\}$ by the end of round $r-1$, i.e., $|I[r]| > t$ (E). Moreover, since p_x proceeds to round $r+1$, $|trusted_x[r]| \geq n-t$ (F)(otherwise, p_x would return $\perp$ at line 08). By combining E and F, we obtain that $\exists y \in I[r] \cap trusted_x[r]$. This means that p_x receives and processes a message from a process p_y, $y \in I[r]$ during round r.

The fact that p_y sends messages during r implies that p_y trusts itself at least until the end of round $r-1$ (line 04). Consequently, p_y takes into account the can_dec sets it has previously computed to update can_dec_y during round $r-1$ (line E07). In particular, as $y \in I[r]$, $y \in can\_dec_y[r_y] \subseteq can\_dec_y[r-1]$. Since $can\_dec_y[r-1]$ is sent by p_y during r, it follows that p_x processes a non empty can_dec set at line E07. Consequently, if $x \in trusted_x[r]$ then, after the

lines E08-E10 have been executed by p_x, we necessarily have $x \in can\_dec_x[r]$ and the lemma follows. $\square_{Lemma\ 8}$

Lemma 9 *Let $i \in Completing[r]$ $(1 \leq r \leq \lfloor \frac{t}{k} \rfloor + 1)$. $can\_dec_i[r] \neq \emptyset \Rightarrow est_i[r]$ is one of the k smallest values in $EST[r]$.*

Proof $can\_dec_i[r] \neq \emptyset$ means that $n-k\ r < |trusted_i[r]|$ (line E09), or p_i has received and processed a message carrying a non-empty can_dec_x set (line E07). We consider each case separately.

- Case 1: $n - k\ r < |trusted_i[r]|$ and each pair (est_x, can_dec_x) received and processed (at lines 09 and E07) by p_i during round r is such that $can\_dec_x = \emptyset$. We claim that, in that case, all the correct processes start round r (*Claim C*).

  If $EST[r] \leq k$, the lemma is trivially correct, so we suppose that $EST[r] > k$. Thanks to Claim C, we can conclude that $\mathcal{C} \subseteq Completing[r-1]$. We can consequently apply Lemma 5 and the lemma follows.

  Proof of the Claim C. We first establish that (assertion A) $\forall r' < r : i \in trusted_i[r'] \wedge can\_dec_i[r'] = \emptyset$. Let us first observe that, as p_i executes $can\_dec_i \leftarrow can\_dec_i \cup \{i\}$ during round r (case assumption), $i \in trusted_i[r]$ (lines E08-E09). It follows then from the management of the $trusted_i$ set (lines 05-07) that (1) $\forall r' < r : i \in trusted_i[r']$. Moreover, as $i \in trusted_i[r]$,p_i receives and processes during r the can_dec set it has computed during $r-1$. Due to the case assumption (i.e., each pair (est_x, can_dec_x) received and processed by p_i during round r is such that $can\_dec_x = \emptyset$), $can\_dec_i[r-1] = \emptyset$. The same reasoning can be applied at round $r-1, \ldots, 1$, from which we conclude that (2) $\forall r' < r : can\_dec_i[r'] = \emptyset$. The assertion follows by combining (1) and (2).

  We have to show that $\mathcal{C} \subseteq Completing[r-1]$. In order to obtain a contradiction, let us suppose that it exists a correct process that stops before the end of round $r-1$. Let r_0 $(\leq r-1)$ be the first round during which a correct process stops and let p_j be a correct process that stops during r_0. This means that either p_i returns $\perp$ at line 08 or p_i decides at line E05. As no correct process decides before r_0 (by the definition of the round r_0), it follows from Lemma 6 that $|trusted_j[r_0]| \geq n - t$, from which we conclude that p_j cannot returns $\perp$ (line 08). Consequently, the only possibility for p_j to stop during r_0 is to decide at line E05. But, in that case, as p_i proceeds to round $r_0 + 1 (\leq r)$ and, due to Lemma 8, we have $i \notin trusted_i[r_0] \vee i \in can\_dec_i[r_0]$. Since $r_0 < r$, this contradicts the assertion A. *End of the proof of the Claim C.*

- Case 2: p_i receives and processes a pair (est_x, can_dec_x) carrying a non-empty can_dec_x set during round r. So, there is a chain of processes $j = j_a, j_{a-1}, \ldots, j_0 = i$ that has carried a non-empty can_dec set to p_i. This chain is such that $a > 0$, $n - k(r-a) < |trusted_j[r-a]|$ is satisfied, and during round $r - x, 0 \leq x \leq a-1$, process j_x receives and processes the pair $(v_{x+1}, can\_dec_{x+1} \neq \emptyset)$ sent by process j_{x+1}. As each process in the chain computes the minimum of the values it has received and processed, $v_{x+1} \geq v_x$ and $v_1 \geq est_i[r]$, where v_1 is the value received by process $j_0 = i$ from process j_1 during r. Hence, $v_a \geq v_1$ where v_a is the value sent by process j_a at round $r - a + 1$. Moreover, at process $j = j_a$, when the predicate $n - k(r-a) < |trusted_j[r-a]|$ is satisfied at round $r-a$, Case 1 applies. Thus, v_a is one of the k smallest value of $EST[r-a]$. Due to Lemma 4, $EST[r] \subseteq EST[r-a]$. Consequently, $v_a \geq est_i[r]$ implies that $est_i[r]$ is one of the k smallest values of $EST[r]$, which proves the lemma for Case 2.

Lemma 10 *Assuming that a process decides at line E05 during round r, let p_x be a process that proceeds to round $r+1$ (if $r = \lfloor \frac{t}{k} \rfloor + 1$, "proceed to round $r+1$" means "execute the* return() *statement at line 11"). We have: $est_x[r]$ is among the k smallest values in $EST[r-1]$.*

Proof Let p_x be a process that proceeds to round $r+1$. Let us observe that the assumptions stated in this lemma and Lemma 8 are the same. Consequently, by using the proof of Lemma 8, we have p_x receives and processes during the round r a pair $(est_y, can\_dec_y \neq \emptyset)$ from a process p_y (this is established in the last paragraph of the proof of Lemma 8).

Let us now consider the value $est_y[r-1]$ sent by p_y to p_x during r. As $y \in Completing[r-1]$ and $can\_dec_y[r-1] \neq \emptyset$, it follows from Lemma 9 that $est_y[r-1]$ is among the k smallest values of $EST[r-1]$. As $est_y[r-1]$ is taken into account by p_x to compute $est_x[r]$ at line 09, we have $est_x[r] \leq est_y[r-1]$. Finally, $EST[r] \subseteq EST[r-1]$ (Lemma 4) allows concluding that $est_x[r]$ is among the k smallest values in $EST[r-1]$, and the lemma follows. $\square_{Lemma\ 10}$

Lemma 11 *Let $r \leq \lfloor \frac{t}{k} \rfloor$ be the first round during which a process decides at line E05. Then, (1) every process that is correct or commits only send omission failures decides at line E05 during round r or $r+1$. Moreover, (2) no process executes more than $r+1$ rounds.*

Proof We assume that a process decides before the end of round $\lfloor \frac{t}{k} \rfloor$. Let r be the first round during which a process decides. Let $DC[r']$ denotes the set of correct processes that decide at line E05 during round r'. Let us notice that, due to the assumption on round number r, $\forall r' < r : DC[r'] = \emptyset$. We first state a claim that follow from the protocol text and the fact a process early decides at r.
Claim C: If a correct process p_c decides during r then, p_c sends messages during r that carry a can_dec_c set such that $c \in can\_dec_c$.
Proof of the Claim C: As $\mathcal{C} \subseteq Completing[r-1]$ (i.e., no correct process decide before the end of round $r-1$), we can apply Lemma 1 from which we obtain $c \in trusted_c[r-1]$. Hence, p_c sends messages (that carry $can\_dec_c[r-1]$) at the beginning of r. Moreover, as p_c decides at line E05 and, due to the test of line E03, we necessarily have $c \in can\_dec_c[r-1]$. *End of the Proof of the Claim C.*

We now prove the lemma by considering two cases:

- First case: every correct process decides during round r.

  Let p_i be a process that commits only send omission failures and does not decide during round r. As $\mathcal{C} \subseteq Completing[r-1]$, it follows from Lemma 6 that $\mathcal{C} \subseteq trusted_i[r]$. This implies that p_i cannot returns $\perp$ at line 08 and then, proceeds to round $r+1$. We now show that $can\_dec_i[r]| > t$. $\mathcal{C} \subseteq trusted_i[r]$ means that p_i receives and processes a message from every correct process during r. Consequently, as every correct p_c sends messages that carry a *can_dec* set that contains c during round r (*Claim C*), $\mathcal{C} \subseteq can\_dec_i[r]$ (line E07). Hence, $|can\_dec_i[r]| > t$. Let us now consider p_i during round $r+1$. Let us first notice that, as p_i adds its identity in the $REC\_FROM_i[r+1]$ set (line E01), $\mathcal{C} \subseteq can\_dec_i[r] \subseteq CAN\_DEC_i[r+1]$ (line E02), which implies that $|CAN\_DEC_i[r+1]| > t$. Moreover, p_i evaluates at $r+1$ the predicate of line E03 to true (Lemma 8). As $|CAN\_DEC_i[r+1]| > t$, p_i decides a value at line E05. This proves the first item of the lemma in the case assumption.

  Let now p_i be a process that commits receive omission failure and does not decide during round r. Suppose that p_i does not decide at line E05 during round $r+1$. Let us remark that,

as every correct process decides during round r, at most t processes send messages during round $r+1$. It follows that the set $rec\_from_i[r+1]$ computed by p_i at line 05 contains at most t process ids, from which we conclude that $\forall j \in rec\_from_i : |W_i(j)| \leq t < n-t$ (line 06). Therefore, $trusted_i[r+1] = \emptyset$ and p_i returns $\perp$ at line 08: the second item of the lemma follows.

- Second case: at least one correct process has not decided at the end of round r.

  Let p_i be a process that is correct or commits only send omission failures. Let us first observe that p_i proceeds to round $r+1$. As r is the first round during which a correct process decides, it follows from Lemma 6 that $\mathcal{C} \subseteq trusted_i[r]$, from which we conclude that p_i cannot returns $\perp$ at line 08 during round r.

  We first show that $DC[r] \subseteq can\_dec_i[r]$. Let us consider a correct process p_c that decides during round r (i.e., $c \in DC[r]$). Due to the claim C, p_c send a messages during r. Moreover, as $\mathcal{C} \subseteq trusted_i[r]$, p_i receives and processes the message sent by p_c during r. Since this message carries a can_dec set such that $c \in can\_dec$ (claim C) , p_i adds c in its can_dec_i set at line E07. This is true for any correct process p_c that decides during round r, hence $DC[r] \subseteq can\_dec_i[r]$.

  We now show that p_i decides at line E05 during round $r+1$. As p_i does not commit send omission failures, p_i receives a message during $r+1$ from every correct process p_c that has not decided during r. Such a message carry a can_dec_c set such that $\{c\} \cup DC[r] \subseteq can\_dec_c$ (the fact that $c \in can\_dec_c$ follows from Lemma 8, as for a correct process p_c we have $c \in trusted_c[r]$). Consequently, it follows from lines E01-E02 that $\mathcal{C} \subseteq CAN\_DEC_i[r+1]$. Moreover, as p_i evaluates the local predicate at line E03 to true (Lemma 8), p_i decides at line E05.

This proves the item (1) of the lemma. As far as item (2) is concerned, let us now consider a faulty process p_i that commits receive omission failure while not crashing. Suppose that p_i does not early decide (at line E05) during rounds r and $r+1$. We show that p_i has returned $\perp$ by the end of round $r+1$.

In order to establish a contradiction, suppose that p_i proceeds to round $r+2$. As p_i does not return $\perp$ at line 08, $|trusted_i[r+1]| \geq n-t$. As $trusted_i[r+1] \subseteq rec\_from_i[r+1] \subseteq REC\_FROM_i[r+1]$, we have $n-t \leq |REC\_FROM_i[r+1]|$. This means that p_i receives messages from at least $n-t$ processes during round $r+1$. Yet, every message sent during $r+1$ carries a can_dec set that contains the id of its sender (Lemma 8). Consequently, $REC\_FROM_i[r+1] \subseteq CAN\_DEC_i[r+1]$ and then, $|CAN\_DEC_i[r+1]| > t$. Finally, as p_i evaluates the predicate of line E03 to true (Lemma 8), p_i decides at line E05: a contradiction. $\square_{Lemma\ 11}$

6.2 Properties of the Protocol

Theorem 4 [Agreement] *No more than k different values are decided.*

Proof To prove the lemma, we consider two cases according to the first round r during which a process decides.

- Case 1: $r \leq \lfloor \frac{t}{k} \rfloor$.
  In that case, any process that decides decides at line E05 during round r or $r+1$ (Lemma 11). We show that any decided value is among the k smallest values in $EST[r-1]$.

Let us first observe that if a process p_i decides at line E05 during a round r', there is a process p_x such that $\exists x \in REC\_FROM_i[r']$ and $can\_dec_x[r'-1] \neq \emptyset$ (possibly $x = i$) (G). This follows from the fact that $|CAN\_DEC_i[r']| > t$ and the way CAN_DEC sets are computed. Moreover, the value $est_x[r-1]$ belongs to $EST_i[r]$ (by the definition of the $EST_i[r]$ set).

- Let p_i be a process that decides during r.

  Let p_x be a process that satisfies the assertion G. As p_x sends messages during r or is p_i itself (this is because $x \in REC\_FROM_i[r]$), $x \in Completing[r-1]$. Since $can\_dec_x[r-1] \neq \emptyset$, it follows from Lemma 9 that $est_x[r-1]$ is among the k smallest values in $EST[r-1]$ (i). Moreover, $est_x[r-1] \in EST_i[r]$ (ii) and, $EST_i[r] \subseteq EST[r-1]$ (i.e., the set $EST_i[r]$ contains only values computed during $r-1$) (iii). Due to the min() function used by p_i to compute the value v that it decides, combining (i), (ii) and (iii) allows concluding that v is among the k smallest values in $EST[r-1]$.

- Let us now consider a process p_i (if any) that decides during round $r+1$.

  As before, let us consider a process p_x as defined in assertion G, so we have $est_x[r] \in EST_i[r+1]$. Since a process has early decided at r and p_x proceeds to round $r+1$ (this is because $x = i$ or, as $x \in REC\_FROM_i[r+1]$, p_x necessarily sends a message during round $r+1$), assumptions of Lemma 10 are satisfied. Consequently, $est_x[r]$ is among the k smallest values in $EST[r-1]$. As $EST_i[r+1] \subseteq EST[r]$ and $EST[r] \subseteq EST[r-1]$ (Lemma 4), we have $EST_i[r+1] \subseteq EST[r-1]$. Moreover, as $est_x[r] \in EST_i[r+1]$, we can conclude that the value decided by p_i is among the k smallest ones in $EST[r-1]$.

- Case 2: $r = \lfloor \frac{t}{k} \rfloor + 1$. We consider two cases according to lines at which processes decides.

  - At least one process decides at line E05. We show that, in that case, any decided value is among the k smallest values in $EST[r-1] (= EST[\lfloor \frac{t}{k} \rfloor])$. Let p_i be a process that decides at line E05 during round r. The reasoning used in the first item of Case 1 is still valid. Consequently, p_i decides one of the k smallest in $EST[r-1]$. Let now p_j be a process that decides at line 11. As a process decides at line E05 of round $r (= \lfloor \frac{t}{k} \rfloor + 1)$ and p_j "proceeds to round $r+1$" (which means here that p_j executes the return() statement at line 11), the assumptions of Lemma 10 are satisfied, from which we conclude that $est_j[r]$ is among the k smallest values in $EST[r-1]$. To conclude, let us observe that p_j decides the value $est_j[r]$.
  - No process decides at line E05. This means that the early decision machinery (i.e., lines E01-E06 and E08-E10) is useless in the considered execution. Let us observe that, if we suppress lines E01-E06 and E08-E10 in the protocol of Figure 2, the resulting protocol is exactly the protocol of Figure 1. Differently said, while no process decides, for all process p_i, the management of variable $trust_i$ and est_i does not differ in the protocols of Figures 1 and 2. This implies that, in the particular execution considered here, we can safely apply Theorem 3 which states that no more than k distinct values are decided.

$\square_{Theorem\ 4}$

Theorem 5 [Strong Termination and Early Stopping] *(i) A process that is correct or commits only send omission failures decides and halts by round* $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$. *(ii) No process halts after* $\min(\lceil \frac{f}{k} \rceil + 2, \lfloor \frac{t}{k} \rfloor + 1)$ *rounds.*

Proof The fact that no process executes more than $\lfloor t/k \rfloor + 1$ rounds is an immediate consequence of the code of the protocol executed in a round-based synchronous model. Moreover, let us observe that the theorem follows directly from Lemma 11 as soon as a process decides at round r such that $r \leq \lfloor f/k \rfloor + 1$. So, to prove the theorem we consider the case where no process decides during a round $\leq \lfloor f/k \rfloor + 1$. Let $f = xk + y \leq t$, where $0 \leq y < k$. (This means that $x = \lfloor f/k \rfloor$.)

Proof of item (i).
Assuming that no no process has decided by round $x+1$, we have to show that that a process that is correct or commits only send omission failures decides and halts by round $\lfloor \frac{f}{k} \rfloor + 2$. To show it, let us consider the consecutive rounds $x+1$ and $x+2$.

- Round $x+1$.
  Let p_i be a process that is correct or commits only send omission failures. We first establish that p_i proceeds to round $x+2$. This is a direct consequence of Lemma 6: as, by assumption, no correct process decides by the end of round $x+1$, $\mathcal{C} \subseteq trusted_i[x+1]$ (A). Then, as p_i does not decides nor crashes during $x+1$, p_i proceeds to round $x+2$.

  We now show that for every process p_i that is correct or commits only send omission failures: $i \in trusted_i[x+1] \Rightarrow i \in can\_dec_i[x+1]$ (B). Let us assume that $i \in trusted_i[x+1]$ and let us consider p_i when it executes lines E07-E10. If $i \in can\_dec_i[x]$ then, as $i \in trusted_i[x+1]$, we still have $i \in can\_dec_i[x+1]$ (line E07). If $i \notin can\_dec_i[x]$, as $i \in trusted_i[x+1]$, p_i evaluates the local predicate of line E09. Moreover, as $\mathcal{C} \subseteq trusted_i[x+1]$ (assertion A), we have $|trusted_i[x+1]| \geq n - f = n - (kx + y) > n - k(x+1)$. Consequently, the predicate is evaluated by p_i to true, from which we conclude (line E09) that $i \in can\_dec_i[x+1]$.

- Round $x+2$.
  Let us first observe that, due to assertion A, every correct process p_c is such that $c \in trusted_c[x+1]$. This implies that p_c sends messages during $x+2$ (line 04). Moreover, due to the assertion B, these messages carry a can_dec_c set such that $c \in can\_dec_c$ (C). Let p_i be a process that is correct or commits only send omission failures. We have to show that p_i decides. As p_i does not commit receive omission failures, it receives from every correct p_c a can_dec_c set such that $c \in can\_dec_c$. Consequently, we have $\mathcal{C} \subseteq CAN\_DEC_i[x+2]$ (lines E01-E02) from which we obtain that $CAN\_DEC_i[x+2] > t$. As p_i evaluates the predicate of line E03 to true (assertion B), p_i decides at line E05. This completes the proof of the first item of the theorem.

Proof of item (ii).
To prove the second item of the theorem (namely no process halts after the round $\lceil \frac{f}{k} \rceil + 2$), we consider two cases. Let us first consider the case where $f = xk + y$ and $y \neq 0$. We have then $\lceil \frac{f}{k} \rceil + 2 = x + 3$. As all the correct processes decide by the end of round $x+2$, the item follows by Lemma 11. The rest of the proof addresses the second case, i.e., $y = 0$.

Let us first observe that assertions A, B and C stated above do not depend on the value of y. We partition the set of correct processes according to the fact they have or not their id in their can_dec set at the end of round x. Let $IC[r]$ denote the subset of correct processes p_c such that $c \in can\_dec_c[r]$ and $\overline{IC[r]}$, the complementary of $IC[r]$ in $\mathcal{C}$ (i.e., $\overline{IC[r]} = \mathcal{C} - IC[r]$). We claim: $|IC[x]| \leq t$ (*Claim C1*) and, $\forall i \in \overline{IC[r]} : trusted_i[x] = \mathcal{C}$ (*Claim C2*).

Proof of the Claim C1. Claim *C1* is obtained by contradiction. Suppose that $|IC[x]| > t$. Let $p_i \in IC[x]$. As no correct process decides by the end of round $x+1$, it follows from Lemma 6 that p_i receives during round $x+1$ a can_dec_c such that $c \in can\_dec_c$ from every process p_c that belongs to

$IC[x]$. Consequently, $IC[x] \subseteq CAN\_DEC_i[x+1]$ (line E01-E02) and then, $t < |CAN\_DEC_i[x+1]|$. Moreover, as $p_i \in IC[x]$ (i.e., $i \in can\_dec_i[x]$) and p_i is correct (i.e., $i \in trusted_i[x]$) p_i evaluates the predicate of line E03 to true, from which we obtain that p_i decides at line E05 during $x+1$: a contradiction. *End of the Proof of the Claim C1.*

Proof of the Claim C2. Let us consider a process $p_i \in \overline{IC[x]}$. Due to Claim *C1*), such a process exists. As p_i does not add its id in $can\_dec_i[x]$ and $i \in trusted_i[x]$ (because p_i is a correct process), p_i evaluates during round x the predicate of line E09 to false. Therefore, $n-kx = n-f \geq |trusted_i[x]|$. As $|\mathcal{C}| = n-f$, it follows from the fact that $\mathcal{C} \subseteq trusted_i[x]$ (assertion A) that $\mathcal{C} = trusted_i[x]$. *End of the Proof of the Claim C2.*

We now establish that (when $y = 0$) $\forall i \in Completing[x+1] : i \in trusted_i[x+1] \Rightarrow i \in can\_dec_i[x+1]$ (B'). This property is true for a process p_i that is correct or commits only send omission failure (assertion B). Let us consider a process p_j that commits receive omission failures. Let us assume that $j \in trust_j[x+1]$. Due to lines 06-07, it follows that p_j receives at least $n-t$ sets $trust[x]$ such that $j \in trust[x]$. This implies that p_j receives and processes at least one $trust_c[x]$ set such that $j \in trust_c[x]$ from a correct process p_c. As p_j is not correct, we necessarily have $\mathcal{C} \subsetneq trusted_c[x]$. Due to Claim *C2*, this process p_c necessarily belongs to $IC[r]$. Hence, p_j also receives from p_c $can\_dec_c[x] \neq \emptyset$ (by definition of $IC[r]$). Consequently, as $j \in trusted_j[x+1]$, it follows from lines E08-E10 that we necessarily have $j \in can\_dec_j[x+1]$.

We now show that a process p_j that commits receive omission failure decides or halts by the end of round $x+2$. In order to establish a contradiction, suppose that p_j proceeds to round $x+3 (= \lceil \frac{f}{k} \rceil + 3)$. In particular, p_j does not return $\perp$ at line 08 during round $x+2$, which means that $|trusted_j[x+2]| \geq n-t$. As $trusted_j[x+2] \subseteq REC\_FROM_j[x+2]$, this implies that p_j receives at least $n-t$ messages during round $x+2$. Moreover, let us observe that, due to assertion B' and the test of line 04, every message sent during $x+2$ carries a can_dec_i such that $i \in can\_dec_i$ (where p_i is the sender). It follows that $|CAN\_DEC_j[x+2]| \geq n-t > t$ (line E01-E02). As p_j uses $trusted_j[x+1]$ and $can\_dec_j[x+1]$ when it executes line E03, due to assertion B', the test is satisfied. It follows that p_j decides at line E05 during round $x+2$: a contradiction. $\square_{Theorem\ 5}$

The next corollary is an immediate consequence of the previous theorem.

Corollary 2 [Termination] *Every correct process decides.*

Theorem 6 [Validity] *A decided value is a proposed value.*

Proof For the processes that decide at line 11, the proof of Theorem 1 applies. So, let us consider a process p_i that decides at line E05. The validity property follows from the fact that $EST_i[r] \subseteq EST[r-1]$ (the values received by p_i during a round r have been determined during the round $r-1$), and $EST_i[r] \neq \emptyset$ (Lemma 7). $\square_{Theorem\ 6}$

Theorem 7 [Bit Complexity] *Let b be the number of bits required to represent a proposed value. The bit complexity is upper bounded by $O(n(b+2n)f/k)$ per process.*

Proof The theorem follows directly from the following observations: (1) at most $\lceil \frac{f}{k} \rceil$ rounds are executed, (2) encoding a set with a bit array, the size of a message sent by a process is $b+2n$, and (3) a process that sends a message sends it to all the processes. $\square_{Theorem\ 7}$

References

[1] Aguilera M.K. and Toueg S., A Simple Bivalency Proof that t-Resilient Consensus Requires $t+1$ Rounds. *Information Processing Letters*, 71:155-178, 1999.

[2] Attiya H. and Welch J., Distributed Computing, Fundamentals, Simulation and Advanced Topics (Second edition). *Wiley Series on Parallel and Distributed Computing*, 414 pages, 2004.

[3] Biran O., Moran S. and Zaks S., A Combinatorial Characterization of the Distributed 1-Solvable Tasks. *Journal of Algorithms*, 11(3): 420-440, 1990.

[4] Biran O., Moran S. and Zaks S., Tight Bounds on the Round Complexity of Distributed 1-Solvable Tasks. *Theoretical Computer Science*, 145(1-2):271-290, 1995.

[5] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, California (USA), pp. 91-100, 1993.

[6] Charron-Bost B. and Schiper A., Uniform Consensus is Harder than Consensus. *Journal of Algorithms*, 51(1):15-37, 2004.

[7] Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.

[8] Chaudhuri S., Herlihy M., Lynch N. and Tuttle M., Tight Bounds for k-Set Agreement. *Journal of the ACM*, 47(5):912-943, 2000.

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

[10] Fischer M.J., Lynch N.A., A Lower Bound on the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183-186, 1982.

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

[12] Gafni E., Guerraoui R. and Pochon B., From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement. *Proc. 37th ACM Symposium on Theory of Computing (STOC'05)*, Baltimore (MD), pp.714-722, May 2005.

[13] Guerraoui R. and Pochon B., The Complexity of Early Deciding Set Agreement: how Topology Can Help? *Proc. 4th Workshop in Geometry and Topology in Concurrency and Distributed Computing (GETCO'04)*, BRICS Notes Series, NS-04-2, pp. 26-31, Amsterdam (NL), 2004.

[14] Hadzilacos V., Issues of Fault Tolerance in Concurrent Computations. *PhD Thesis, Tech Report 11-84*, Harvard University, Cambridge (MA), 1985.

[15] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993.

[16] Herlihy M.P. and Penso L. D., Tight Bounds for k-Set Agreement with Limited Scope Accuracy Failure Detectors. *Distributed Computing*, 18(2): 157-166, 2005.

[17] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.

[18] Keidar I. and Rajsbaum S., A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters*, 85:47-52, 2003.

[19] Lamport L. and Fischer M., Byzantine Generals and Transaction Commit Protocols. *Unpublished manuscript*, 16 pages, April 1982.

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

[21] Mostéfaoui A. and Raynal M., k-Set Agreement with Limited Accuracy Failure Detectors. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, Portland (OR), 2000.

[22] Mostéfaoui A. and Raynal M., Randomized Set Agreement. *Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01)*, ACM Press, pp. 291-297, Hersonissos (Crete), 2001.

[23] Neiger G. and Toueg S., Automatically Increasing the Fault-Tolerance of Distributed Algorithms. *Journal of Algorithms*, 11:374-419, 1990.

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

[25] Perry K.J. and Toueg S., Distributed Agreement in the Presence of Processor and Communication Faults. *IEEE Transactions on Software Eng.*, SE-12(3):477-482, 1986.

[26] Raïpin Parvédy Ph. and Raynal M., Optimal Early Stopping Uniform Consensus in Synchronous Systems with Process Omission Failures. *Proc. 16th ACM Symposium on Parallel Algorithms and Architectures (SPAA'04)*, Barcelona (Spain), ACM Press, pp. 302-310, 2004.

[27] Raïpin Parvédy Ph., Raynal M. and Travers C., Early-Stopping k-set Agreement in Synchronous Systems Prone to any Number of Process Crashes. *8th Int'l Conference on Parallel Computing Technologies (PaCT'05)*, Krasnoyarsk (Russia), Springer Verlag LNCS #3606, pp. 49-58, 2005.

[28] Raïpin Parvédy Ph., Raynal M. and Travers C., Decision Optimal Early-Stopping k-set Agreement in Synchronous Systems Prone to Send Omission Failures. *Proc. 11th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'05)*, Changsa (China), IEEE Computer Press, pp. 23-30, 2005.

[29] Raïpin Parvédy Ph., Raynal M. and Travers C., Strongly Terminating Early-Stopping k-set Agreement in Synchronous Systems with General Omission Failures. *Proc. 13th Colloquium on Structural Information and Communication Complexity (SIROCCO'06)*, Springer-Verlag LNCS #4056, pp. 182-196, Liverpool (UK), 2006.

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

[31] Raynal M. and Travers C., Synchronous Set Agreement: a concise guided tour (including a new algorithm and a list of open problems). *Proc. 12th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'05)*, Riverside (CA), IEEE Computer Press, 2006.

[32] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.

[33] Yang J., Neiger G. and Gafni E., Structured Derivations of Consensus Algorithms for Failure Detectors. *Proc. 17th Int'l ACM Symposium on Principles of Distributed Computing (PODC'98)*, ACM Press, pp. 297-308, Puerto Vallarta (Mexico), July 1998.