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

Philippe Raïpin Parvédy, Michel Raynal, and Corentin Travers

IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, 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 failures 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 very interesting consequences. First, it shows that, although 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 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.

P. Flocchini and L. Gąsieniec (Eds.): SIROCCO 2006, LNCS 4056, pp. 182–196, 2006.

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, 31]. (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, 32]).

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,

[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 $f = t$.

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).

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 (contrarily to what we could "normally" hope) 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].

Content of the paper. This paper investigates the k-set agreement problem in synchronous systems prone to general omission failures and presents a corresponding protocol. 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 input or output buffer overflow 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 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). 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[3].

– 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 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 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.

[3] 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.

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].)

k-set protocol can be useful to allocate shareable 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 a 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. Due to the page limitation, the full proofs of these properties do not appear in this paper. The interested reader can find them in a companion technical report [29] available on-line.

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, *good* processes are the processes that neither crash nor commit 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. Its initial value is v_i (line 01).
– $trusted_i$ represents the set of processes that p_i currently considers as being correct. Its 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".

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.

```
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)
```

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

- 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 full proof of the protocol is given in [29]. The protocol proof assumes $t < n/2$. It uses the following notations.

- Given a set of process identities $X = \{i, j, \ldots\}$, 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: $Completing[r+1] \subseteq Completing[r], Silent[r] \subseteq Silent[r+1], \forall i \in Completing[r] : \ Silent[r] \subseteq \Pi - trusted_i[r]$.

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) $i \in Completing[r]$.*

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[4]) 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. $\forall r \geq 1 : \forall x : \left|\{y : y \in Completing[r-1] \wedge x \in trusted_y[r-1]\}\right| < n - t \Rightarrow x \in Silent[r]$.

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.* $\forall r \geq 1 : \left|\{x : est_x[r-1] \leq v \ \wedge \ x \in Completing[r-1]\}\right| \geq n - t \Rightarrow \forall i \in Completing[r] : est_i[r] \leq v$.

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

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

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

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 $n - k\,r < |trusted_i[r]| \Rightarrow est_i[r] \leq v_m$.*

4.3 Properties of the Protocol

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

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

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.*

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.

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 $\emptyset$.

```
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) rec. 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) rec. 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)then
(E09)     if (n − k r < |trusted_i|) ∨ (can_dec_i ≠ ∅) then can_dec_i ← can_dec_i ∪ {i}
(E10)   endif  endif
(10)  end_round;
(11)  return (est_i)
```

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

- 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 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

Detailed proofs of the following lemmas and theorems are given in [29].

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) $i \in Completing[r]$.*

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.*

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: $x \notin trusted_x[r] \vee x \in can\_dec_x[r]$.*

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 $EST[r]$.*

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 $EST[r-1]$.*

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.*

6.2 Properties of the Protocol

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

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.*

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.*

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.*

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. 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. 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. *Tech Report #1711*, IRISA, Université de Rennes (France), 22 pages 2005.
ftp://ftp.irisa.fr/techreports/2005/PI-1711.ps.gz
30. Raynal M., Consensus in Synchronous Systems: a Concise Guided Tour. *Proc. 9th IEEE Pacific Rim Int. Symposium on Dependable Computing (PRDC'02)*, Tsukuba (Japan), IEEE Computer Press, pp. 221-228, 2002.
31. 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.
32. Yang J., Neiger G. and Gafni E., Structured Derivations of Consensus Algorithms for Failure Detectors. *Proc. 17th Int. ACM Symposium on Principles of Distributed Computing (PODC'98)*, ACM Press, pp. 297-308, Puerto Vallarta (Mexico), July 1998.