# Synchronous Set Agreement: a Concise Guided Tour (including a new algorithm and a list of open problems)

Michel RAYNAL Corentin TRAVERS

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

## Abstract

*The $k$-set agreement problem is a paradigm of coordination problems encountered in distributed computing. The parameter $k$ defines the coordination degree we are interested in. (The case $k = 1$ corresponds to the well-known uniform consensus problem.) More precisely, the $k$-set agreement problem considers a system made up of $n$ processes where each process proposes a value. It requires that each non-faulty process decides a value such that a decided value is a proposed value, and no more than $k$ different values are decided.*

*This paper visits the $k$-set agreement problem in synchronous systems where up to $t$ processes can experience failures. Three failure models are explored: the crash failure model, the send omission failure model, and the general omission failure model. Lower bounds and protocols are presented for each model. Open problems for the general omission failure model are stated. This paper can be seen as a short tutorial whose aim is to make the reader familiar with the $k$-set agreement problem in synchrony models with increasing fault severity. An important concern of the paper is simplicity. In addition to its survey flavor, several results and protocols that are presented are new.*

## 1 Introduction

**Coordination problems and $k$-set agreement** Coordination problems are central in the design of distributed systems where processes have to exchange information and synchronize in order to agree in one way or another (for otherwise, they would behave as independent Turing machines, and the system would no longer be a distributed system). This paper surveys one distributed coordination problem, namely, the $k$-set agreement problem. This survey focuses on recent results in synchronous systems.

The *$k$-set agreement* problem has been introduced in [4]. It can can be defined as follows. Considering a system made up of $n$ processes where each process proposes a value, and up to $t$ processes can experience failures, 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. The well-known consensus problem is nothing else than the 1-set agreement problem, where the non-faulty process have to decided the same value. The parameter $k$ of the set agreement can be seen as the degree of coordination associated with the corresponding instance of the problem. The smaller $k$, the more coordination among the processes: $k = 1$ means the strongest possible coordination, while $k = n$ means no coordination.

Be the message-passing system synchronous or asynchronous, the *$k$-set agreement* problem can always be solved despite process crash failures, as soon as $k > t$. A trivial solution is as follows: $k$ predefined processes broadcast their value to all the processes, and a process simply decides the first value it receives. (It is easy to see that, whatever the crash pattern, at most $k$ values are sent, and at least one value is sent.)

**$k$-set agreement solvability** Surprisingly, while the $k$-set agreement can be trivially solved in asynchronous systems when the coordination degree $k$ is such that $k > t$, there is no deterministic protocol that can solve it in such a system as soon as $k \leq t$ (e.g., see [12] for such an impossibility proof). This impossibility result generalizes the impossibility to solve the consensus problem in asynchronous systems where even only one process can crash (case $k = t = 1$) [8]. This means that solving the $k$-set agreement problem in an asynchronous system requires either to enrich this system with additional power (such as the one provided by failure detectors [11, 16] or random numbers [17]), or restrict the input vectors that the processes can collectively propose [3, 15].

Differently, the $k$-set agreement problem can always be solved in synchronous systems prone to process crash failures. The protocols that solves it are all based on the *round*

12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)
0-7695-2724-8/06 $20.00 © 2006 IEEE

notion. The processes execute a sequence of rounds and, during each round, each process executes sequentially the following steps: it first sends messages, then receives messages, and finally executes local computation. The main property of a synchronous system is that the messages sent during a round are received during the same round.

**$k$-set agreement efficiency** A fundamental question associated with $k$-set agreement concerns the minimal number of rounds that any protocol has to execute in the worst case scenario where up to $t$ processes crash (the time complexity of a synchronous protocol is usually measured as the maximal number of rounds it requires). It has been shown that $lb_t = \lfloor \frac{t}{k} \rfloor + 1$ is a lower bound on that number of rounds. This means that, whatever the $k$-set agreement protocol, it is always possible to have a run of that protocol that requires at least $lb_t$ rounds for the processes to decide [5]. (This worst case scenario is when exactly $k$ processes crash during each round, in such a way that -during the round in which it crashes- a process sends values to some non-crashed processes but not to all of them.) It is important to notice that the previous bound states an "inescapable trade-off" relating the fault-tolerance parameter $t$, the degree $k$ of coordination achieved, and the best time complexity $lb_t$ that a set agreement protocol can attain [5]. Moreover, it is worth noticing that, when compared to the consensus problem, $k$-set agreement divides the time by $k$.

Another fundamental question concerns the *adaptivity* of a $k$-set agreement protocol to the "good" runs. Those are the runs where there are few crashes, i.e., the runs where the number of actual crashes $f$ is smaller than $t$ (the maximum number of crashes for which the protocol works). This is the notion of *early decision* [6]. It has very recently been shown that there is no $k$-set agreement protocol that, in presence of $f$ process crashes, allows the processes to always decide before $lb_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds [9]. This bound shows an additional relation linking the best time efficiency a set agreement protocol can attain, the actual number of crashes $f$, and the coordination degree $k$. It is worth noticing that, in failure-free runs ($f = 0$), two rounds are sufficient for the processes to coordinate (decide) whatever the value of $k$ (which is the lower bound for solving the uniform consensus problem in failure-free runs [13]).

**Content of the paper** The spirit of this paper is the same as [24] (devoted to consensus). The paper visits $k$-set agreement protocols in several process failure models, namely, the classical crash failure model, the send omission failure model and the general omission failure model. In the send omission failure model, during a round, a process can crash or forget to send messages. In the general omission failure model, a process can additionally forget to receive messages. In addition to the (five) protocols that are described, it is shown that the previous lower bounds $lb_t$ and $lb_f$ are still valid for the send omission failure model for any value of $t$ (i.e., $t < n$), and for the general omission failure model when $t < n/2$. A new protocol for general omission failures that works for $t < \frac{k}{k+1}n$ is also presented. It is shown that this protocol is optimal with respect to the resilience bound $t$.

The paper also introduces a new property for the $k$-set agreement problem in presence of omission failures. This property, called *strong termination*, requires that the processes that commit only send omission failures (and do not crash) decide as if they were non-faulty. This allows more processes to decide. Open problems that concern $k$-set agreement in the general omission failure models are also presented. A main accent of the paper is simplicity. (Due to page limitation, it was not possible to include the proofs of the protocols and the theorems presented in the paper. They can be found in [25].) In that sense, the paper can be considered as an "introductory survey". As indicated in the abstract, this paper can be seen as a short tutorial whose aim is to make the reader familiar with the $k$-set agreement problem in synchrony models with increasing fault severity. In addition to its survey flavor, several results and protocols that are presented are new. Moroever, some problems that remain open are stated.

## 2 Distributed Computing Model

### 2.1 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 reliable channel (which means that 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. 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 received.
- Send Omission failure. A faulty process crashes or omits sending messages it was supposed to send.
- 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) [19].

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.

A send (receive) omission failure actually models a failure of the output (input) buffer of a process. A buffer overflow is a typical example of such a failure. An intuitive explanation of the fact that it is more difficult to cope with receive omission failures than with send omission failures is the following. A process that commits only send omission failure continues to receive the messages sent by the correct process. Differently, when a process commits receive omission, it experiences an "autism" behavior.

## 3 The $k$-Set Agreement Problem

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 set agreement problem with coordination degree $k$, 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 $k > t$).

```
Function set_agreement (v_i)
    est_i ← v_i;
    when r = 1,2,... , ⌊t/k⌋ + 1 do % r: round number %
    begin_round
        send (est_i) to all; % including p_i itself %
        est_i ← min({est_j values rec. during the current round r})
    end_round;
    return (est_i)
```

**Figure 1. Crash failures: Synchronous $k$-set agreement, code for $p_i$ ($t < n$)**

## 4 Set Agreement in the Crash Failure Model

### 4.1 A simple protocol

A very simple synchronous $k$-set agreement protocol for the most general crash failure model (i.e., $t < n$) is described in Figure 1 (this is the classical protocol presented in distributed computing textbooks (e.g., [2, 14]). A process $p_i$ invokes the protocol by calling the function set_agreement $(v_i)$ where $v_i$ is the value it proposes. If it does not crash, $p_i$ terminates when it executes the return() statement.

The idea is for a process to decide the smallest estimate value it has ever seen. To attain this goal, the protocol is based on the flooding strategy. Each process $p_i$ maintains a local variable $est_i$ that contains its current estimate of the decision value. Initially, $est_i$ is set to $v_i$ the value proposed by $p_i$. Then, during each round, each non-crashed process first broadcasts its current estimate, and then updates it to the smallest values among the estimates it has received. (The proof of this protocol appears as a particular case of the proof of the early-deciding protocol that follows.)

### 4.2 An early-deciding protocol

An early deciding $k$-set agreement protocol for the crash failure model is presented in Figure 2. This protocol (introduced in [21]) is a generalization of the previous flood-set protocol. Its underlying principles are the following. Let $nb_i[r]$ be the number of processes from which a process $p_i$ has received messages during the round $r$ (by definition, $nb_i[0] = n$). As crashes are stable (there is no recovery), we have $nb_i[r-1] \geq nb_i[r]$.

This simple observation incite investigating the local predicate $nb_i[r-1] - nb_i[r] < k$. When true, this predicate means that $p_i$ is missing the current estimates from at most $k-1$ processes among all the processes that were alive at the beginning of the round $r$. Combined with the systematic use of the flooding strategy, this allows $p_i$ to conclude that it knows one of the $k$ smallest value present in the system.

Unfortunately, the local predicate $nb_i[r-1] - nb_i[r] < k$ is not powerful enough to allow $p_i$ to also conclude that the

```
Function set_agreement (v_i)
  est_i ← v_i; nb_i[0] ← n; can_dec_i ← false;
  when r = 1, 2, ..., ⌊t/k⌋ + 1 do % r: round number %
  begin_round
     send (est_i, can_decide_i) to all; % including p_i itself %
**   if can_decide_i then return (est_i) end_if;
     let nb_i[r] = number of messages received by p_i during r;
     let decide_i = ∨ on the can_decide_j boolean values rec. during r;
     est_i ← min({est_j values rec. during the current round r});
     if ((nb_i[r − 1] − nb_i[r] < k) ∨ decide_i)
                  then can_decide_i ← true end_if
  end_round;
  return (est_i)
```

**Figure 2. Early stopping synchronous $k$-set agreement: code for $p_i$ ($t < n$)**

other processes know it has one of the $k$ smallest values. Consequently, $p_i$ cannot decide and stop immediately. To be more explicit, let us consider the case where the current estimate of process $p_i$ is the smallest value $v$ in the system, $p_i$ is the only process that knows $v$, $p_i$ decides $v$ at the end of $r$, and crashes immediately after deciding. The other processes can then decide $k$ other values as $v$ is no longer is the system from round $r + 1$. An easy way to fix this problem consists in requiring $p_i$ to proceed to the round $r + 1$ before deciding. When $nb_i[r-1] - nb_i[r| < k$ becomes true, $p_i$ sets a boolean ($can\_decide_i$) to $true$ and proceeds to the next round $r + 1$. As, before deciding at line ** of $r + 1$, $p_i$ has first sent the pair $(est_i, can\_decide_i)$ to all processes, any process $p_j$ active during $r + 1$ not only knows $v$ but, as $can\_decide_i$ is true, knows also that $v$ is one of $k$ smallest values present in the system during $r + 1$. The protocol follows immediately from these observations.

The protocol is early-deciding, namely, a process that does not crash decides at the latest during the round $lb_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds. Its correctness proof can be found in [21].

### 4.3 On the early decision predicate

Instead of using the local predicate $nb_i[r-1] - nb_i[r] < k$, an early stopping protocol could be based on the local predicate $faulty_i[r] < k\ r$ where $faulty_i[r] = n - nb_i[r]$ (the number of processes perceived as faulty by $p_i$)[1]. While both predicates can be used to ensure early stopping, we show here that $nb_i[r-1] - nb_i[r] < k$ is a more efficient predicate than $faulty_i[r] < k\ r$ (more efficient in the sense that it can allow for earlier termination). To prove it, we show the following:

- $(i)$ Let $r$ be the first round during which the local predicate $faulty_i[r] < k\ r$ is satisfied. The predicate $nb_i[r-1] - nb_i[r] < k$ is then also satisfied.
- $(ii)$ Let $r$ be the first round during which the local predicate $nb_i[r-1] - nb_i[r] < k$ is satisfied. It is possible that $faulty_i[r] < k\ r$ be not satisfied.

We first show $(i)$. As $r$ is the first round during which $faulty_i[r] < k\ r$ is satisfied, we have $faulty_i[r-1] \geq k\ (r-1)$. So, we have $faulty_i[r] - faulty_i[r-1] < k\ r - k\ (r-1) = k$. Replacing the sets $faulty_i[r]$ and $faulty_i[r-1]$ by their definitions we obtain $(n - nb_i[r]) - (n - nb_i[r-1]) < k$, i.e., $(nb_i[r-1] - nb_i[r]) < k$.

A simple counter-example is sufficient to show $(ii)$. Let us consider a run where $f1 > ak$ ($a > 2$) processes crash initially (i.e., before the protocol starts), and $f2 < k$ processes crash thereafter. We have $n - f1 \geq nb_i[1] \geq nb_i[2] \geq n - (f1 + f2)$, which implies that $(nb_i[r-1] - nb_i[r]) < k$ is satisfied at round $r = 2$. On an other side, $faulty_i[2] \geq f1 = ak > 2k$, from which we conclude that $faulty_i[r] < r\ k$ is not satisfied at $r = 2$.

This discussion shows that, while the early decision lower bound can be obtained with any of these predicates, the predicate $nb_i[r-1] - nb_i[r] < k$ is more efficient in the sense it takes into consideration the actual failure pattern (a process counts the number of failures it perceives during each round, and not only from the beginning of the run). Differently, the predicate $faulty_i[r] < r\ k$ considers only the actual number of failures and not their pattern (it basically always considers the worst case where there are $k$ crashes per round, whatever their actual occurrence pattern).

## 5 Set Agreement in the Send Omission Model

Let us now consider that, in addition to crash, a process can also fails by omitting to send messages. This means that, during a round, a process can send a message to some processes and forget to send a message to some other processes. Similarly to the crash failure model, the fact that a process $p_i$ does not receive a message from $p_j$ can allow $p_i$ to conclude that $p_j$ is faulty, but differently, it cannot allow it to conclude that $p_j$ has crashed.

### 5.1 A simple protocol

A simple $k$-set agreement protocol that can cope with up to $t < n$ faulty processes is described in Figure 3. This protocol (that is a variant of a protocol introduced in [10]) is obtained from the previous basic flooding protocol by two simple modifications. They concern the starred lines.

The underlying idea is the following one. During a round $r$ ($1 \leq r \leq \lfloor t/k \rfloor + 1$), only the processes $p_i$ such that $(r-1)k < i \leq rk$ send their estimate values. As far as

[1] This predicate is implicitly used in the proof of the (not-early deciding) $k$-set agreement protocol described in [14].


0-7695-2724-8/06 $20.00 

```
Function set_agreement (v_i)
    est_i ← v_i;
    when r = 1, 2, ..., ⌊t/k⌋ + 1 do % r: round number %
    begin_round
(*1)  if (i is such that (r − 1)k < i ≤ rk)
                then send (est_i) to all end_if;
(*2)  est_i ← any est_j rec. during r if any, unchanged otherwise
    end_round;
    return (est_i)
```

**Figure 3. Send omission failures: Synchronous $k$-set agreement, code for $p_i$ ($t < n$)**

message reception is concerned, at the end of a round, a process $p_i$ defines its estimate $est_i$ as being any estimate value it has received during that round. If it has received no estimate, $est_i$ keeps its previous value.

The protocol is based on the following simple principle: restricting each round to have at most $k$ senders. As $(\lfloor t/k \rfloor + 1)k > t$ and at most $t$ processes crash or commit send omission failures, there is at least one round (say $R$) that has a correct sender $p_c$. This means that during $R$, all the processes receives an estimate from $p_c$. Consequently, any non-crashed process updates its estimate during $R$. Finally, at most $k$ different estimates can be adopted during a round (line *2). It follows that, from round $R$, there are at most $k$ distinct values in the system. Interestingly, this protocol associates specific senders with each round (which, in some sense, means that it forces the other processes to simulate send omission failures during that round).

### 5.2 Early decision and strong termination

An early-deciding $k$-set protocol for the send omission failure model with $t < n$, is described in [22]. No process decides after the round $lb_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$. Thanks to this protocol, we have the following theorem.

**Theorem 1** *$lb_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ is a lower bound on the number of rounds for solving $k$-set agreement in the synchronous send omission failure model with $t < n$.*

**Proof** The theorem follows from the following observations. (1) The very existence of the previous early-deciding protocol. (2) The fact that $lb_f$ is a lower bound in the crash failure model. And, (3) the fact that the send omission failure model includes (is more severe than) the crash failure model. $\square_{Theorem\ 1}$

In addition to the termination, validity and agreement properties that defines the $k$-set agreement problem, the early deciding protocol described in [22] enjoys the following noteworthy property:

- Strong termination: a process that does not crash decides.

It is worth noticing that each of these properties (early decision vs strong termination) is not obtained at the detriment of the other. Strong termination is a property particularly meaningful when one is interested in solving agreement problems despite omission failures. Intuitively, it states that a protocol has to force as many processes as possible to decide.

**Problem difficulty** The non-early deciding protocol described in Figure 3 and the early deciding protocol described in [22] shows that the $k$-set agreement problem has the same lower bounds in the crash failure model and the send omission failure model. This means that, for that problem, the send omission failure model is not "more difficult" than the crash model. As we are about to see, this is no longer true for the general omission failure model.

## 6 Set Agreement in the General Omission Failure Model when $t < n/2$

Let us now consider the more severe failure model where a process can crash, omit to send messages or omit to receive messages. We first address the case where no more than $t < n/2$ processes can be faulty. This section presents an optimal $k$-set agreement protocol suited to this context and states an open problem.

### 6.1 A strongly terminating protocol for $t < n/2$

There is no way to force a process that commits receive omission failures to decide one of the $k$ values decided by the other processes. This is because, due to its faults, such a process can never know these values. In that case, the protocol forces such processes to stop without deciding a value (let us remind that the problem requires "only" that the correct processes decide). A faulty process that does not decide, returns a default value denoted $\perp$ whose meaning is "no decision" from the $k$-set agreement point of view. By a language abuse we then say that such a process "decides $\perp$".

**Local variables** The protocol, described in Figure 4, has been proposed in [23]. In addition to $est_i$, a process $p_i$ manages three local variables whose meaning is the following:

- $trusted_i$ represents the set of processes that $p_i$ currently considers as being correct. It initial value is $\Pi$ (the whole set of processes). So, $i \in trusted_i$ (line 4) 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 $p_j$".)

```
Function set_agreement(v_i)
(1)  est_i ← v_i; trusted_i ← Π;   % r = 0 %
(2)  for r = 1,..., ⌊t/k⌋ + 1 do
(3)  begin_round
(4)    if (i ∈ trusted_i) then for_each j ∈ Π do send(est_i, trusted_i) to p_j end_do end_if;
(5)    let rec_from_i = {j : (est_j, trust_j) is received from p_j during r ∧ j ∈ trusted_i};
(6)    for_each j ∈ rec_from_i let W_i(j) = {ℓ : ℓ ∈ rec_from_i ∧ j ∈ trust_ℓ};
(7)    trusted_i ← rec_from_i − {j : |W_i(j)| < n − t};
(8)    if (|trusted_i| < n − t) then return (⊥) end_if;
(9)    est_i ← min(est_j received during r and such that j ∈ trusted_i)
(10) end_round;
(11) return (est_i)
```

**Figure 4. General omission failures: strongly terminating $k$-set protocol, code for $p_i$ ($t < \frac{n}{2}$)**

- $rec\_from_i$ 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 5).
- $W_i(j)$ is a set of process identities associated with the processes $p_\ell$ that are currently trusted by $p_i$ and that (to $p_i$'s knowledge) trust $p_j$ (line 6). ($W_i(j)$ stands for "Witness of $p_j$".)

**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 "some subset" of values it has received. The crucial part of the protocol consists in providing each process with correct rules that allow it to determine a "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.

- 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 4). Otherwise, $p_i$ skips the sending phase.
- Then, $p_i$ executes the receive phase (line 5). As already indicated, when it considers the messages it has received during the current round, $p_i$ considers only the messages sent by 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 6-9). 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 6-7). 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 8). If the answer is negative, $p_i$ discovers 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 8, $p_i$ computes its new estimate of the decision value (line 9) according to the estimate values it has received from the processes it currently trusts.

A proof of this protocol can be found in [23]. The role of the $W_i(j)$ control variable and the associated predicate $|W_i(j)| < n-t$ are central to ensure the strong termination property. Let $p_i$ be a faulty process that neither crashes, nor commits receive omission failures (i.e., it commits only send omission failures). Let us observe that, at each round, such a $p_i$ receives a message from each correct process $p_j$. This means that, with respect to each correct process $p_j$, we always have $|W_i(j)| \geq n - t$ (lines 6-7). Consequently, $p_i$ always trusts all correct processes, and so we always have $|trusted_i| \geq n - t$. It follows that such a process $p_i$ cannot stop at line 8, and decides consequently at line 11.

## 6.2 A first open problem

As far as early-decision is concerned, to our knowledge, only one protocol has been designed for the general omission failure model with $t < n/2$. This protocol (introduced in [23]) enjoys the following properties:

- It is strongly terminating.
- Any process that commits only send omission failures (and does not crash) decides in at most $lb_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$, which shows that this is a lower bound on the time complexity for the $k$-set agreement

problem in the general omission failure model where $t < n/2$.

- 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.

The following problem remains open: Is $\lceil \frac{f}{k} \rceil + 2$ a tight lower bound for a process that commits receive omission failure (and does not crash) to stop when $f = k\ x + y$, where $x$ and $y$ are integers such that $0 < y < k$.

# 7 Set Agreement in the General Omission Failure Model when $t \geq n/2$

Let us finally consider the general omission failure model when the "majority of correct processes" assumption is no longer valid. This section first shows that there is no $k$-set agreement protocol that can cope with general omission failures when $t \geq \frac{k}{k+1}n$. Then, it presents a protocol showing that $t < \frac{k}{k+1}n$ is a tight lower bound. Finally, problems are stated, that remain open in the general omission failure model.

## 7.1 A resilience bound for $k$-set agreement

Several $k$ set agreement protocols have been designed for the crash failure and the send omission failure models. They all consider the most general case, i.e., $t < n$. Very differently, to our knowledge, only one $k$-set protocol has been designed for the general omission failure model (the protocol presented in the previous section [23]), and that protocol considers $t < n/2$. (Several protocols have been designed for the particular case $k = 1$ -consensus problem-, e.g., [18, 20, 24], where it is shown that $t < n/2$ is an upper bound on the value of $t$). So the following fundamental question comes immediately to mind: Does $t < n/2$ define the upper bound for the value of $t$ when one is interested in solving the $k$ set agreement problem, for any $k \geq 1$? This section shows that it is not. The lower bound on the maximal number of faulty processes is $t < \frac{k}{k+1}n$. The next subsection shows that this lower bound it tight by providing a corresponding protocol.

**Theorem 2** *There is no $k$-set agreement protocol in synchronous systems prone to general omission failures when $t \geq \frac{k}{k+1}n$.*

The proof is a straightforward generalization of proofs that show there is no uniform consensus protocol in synchronous systems prone to general omission failures when $t \geq n/2$ [18, 24]. It is based on a (simple) classical partitioning argument. Due to page limitation it is given in [25].

## 7.2 A protocol for $t < \frac{k}{k+1}n$

This section presents a new, yet very simple, protocol that solves the $k$-set agreement problem despite up to $t$ processes that commit general omission failures in a synchronous system where $t < \frac{k}{k+1}n$. To our knowledge, the design of such a protocol has never been addressed before. This protocol requires $t - k + 2$ rounds. To make more visible the meaning of this number, it can be rewritten as $(t + 1) - (k - 1)$. It is easy to see that for $k = 1$, this is the $t + 1$ consensus lower bound, and $t < \frac{k}{k+1}n$ becomes $t < n/2$, which is a necessary condition for that problem in the general omission failure setting (see the "Open problems" section that follows).

```
Function set_agreement(v_i)
(1)  est_i ← v_i; trusted_i ← Π;   % r = 0 %
(2)  for r = 1, ..., [t + 2 − k] do
(3)  begin_round
(4)    for_each j ∈ trusted_i do send est_i to p_j end_do;
(5)    foreach j ∈ trusted_i do
(6)      if (est_j received from p_j) then est_i ← min(est_i, est_j)
(7)                                   else trusted_i ← trusted_i − {j}
(8)      end_if
(9)    end_do;
(10)   if (|trusted_i| < n − t) then return (⊥) end_if;
(11) end_round;
(12) return (est_i)
```

**Figure 5. $k$-set protocol for general omission failures, code for $p_i$ ($t < \frac{k}{k+1}n$)**

Differently from its proof that is not trivial (see [25] for a proof), the design of this protocol is particularly simple. It is similar to the early-deciding uniform consensus protocol presented in [20] from which the early decision part is suppressed. More precisely, the protocol can be seen as managing two variables, a control variable (a set denoted $trusted_i$ that contains the processes it considers as non-faulty), and a data variable, namely, its current estimate $est_i$. More specifically, we have the following:

- A process $p_i$ sends its current estimate only to the processes in $trusted_i$, and accept receiving estimates only from them. Basically, it communicates only with the processes it trusts (lines 4-9). In that way, if during a round $r$, $p_j$ commits a send omission failure with respect to $p_i$, or if $p_i$ commits a receive omission failure with respect to $p_j$, $p_i$ and $p_j$ will not trust each other from the round $r + 1$. Interestingly, this ensures that, if $p_i$ is correct, it will always trust at least $n - t$ processes. So, if during a round $r$, a process finds that it trusts less than $n - t$ processes, it can conclude that it is faulty, and consequently decides the default value $\perp$ (line 10).

- Each data local variable $est_i$ is used as in the previous protocols. It contains the smallest value that $p_i$ has ever received from the processes it currently trusts.

This simple management of the variables $trusted_i$ and $est_i$, solves the $k$-set agreement problem despite up to $t < \frac{k}{k+1}n$ processes prone to general omission failures. This protocol is proved in [25]. It is not strongly terminating.

### 7.3 Four more open problems

Concerning the $k$-set agreement problem in synchronous systems where up to to $t < \frac{k}{k+1}n$ processes can commit general omission failures, four problems (at least) remain open.

- Is $t - k + 2$ a lower bound on the number of rounds? (Let us remind that $t + 1$ is the lower bound for the consensus problem [1, 7], i.e., when $k = 1$.)
- How to design an early-deciding protocol? Which is the corresponding early-deciding lower bound?
- Is it possible to design a strongly terminating protocol? If the answer is "yes", design such a protocol.
- Is there a proof simpler than the one described in [25], for the protocol described in Figure 5.

These questions remain open challenges for people interested in synchronous agreement problems and lower bounds.

## 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 Par. and Dist. Computing*, 414 pages, 2004.

[3] Attiya H. and Avidor Z., Wait-Free $n$-Set Consensus when Inputs are Restricted. *Proc. 16th Int. Symposium on Distributed Computing (DISC'02)*, Springer Verlag LNCS #2508, pp. 326-338, 2002.

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

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

[6] Dolev D., Reischuk R. and Strong R., Early Stopping in Byzantine Agreement. *JACM*, 37(4):720-741, April 1990.

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

[8] 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.

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

[10] Guerraoui R., Kouznetsov P. and Pochon B., A note on Set Agreement with Omission Failures. *Electronic Notes in Theoretical Computer Science*, Vol. 81, 2003.

[11] 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.

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

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

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

[15] Mostéfaoui A., Rajsbaum S. and and Raynal M., The Combined Power of Conditions and Information on Failures to Solve Asynchronous Set Agreement. *Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC'05)*, ACM Press, pp. 179-188, 2005.

[16] 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, 2000.

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

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

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

[20] 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*, pp. 302-310, 2004.

[21] 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)*, Springer Verlag LNCS #3606, pp. 49-58, 2005.

[22] 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)*, IEEE Computer Press, pp. 23-30, 2005.

[23] 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, 2006.

[24] 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.

[25] Raynal M. and Travers C., Synchronous Set Agreement: a Concise Guided Tour (with open problems). *Tech Report #1791*, IRISA, Université de Rennes (France), 22 pages, 2006. (www.irisa.fr/bibli/publi/pi/2006/1791/1791.html)