Synchronous Set Agreement: a Concise Guided Tour (with open problems)

Michel Raynal, Corentin Travers

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1791

SYNCHRONOUS SET AGREEMENT: A CONCISE GUIDED TOUR (WITH OPEN PROBLEMS)

MICHEL RAYNAL CORENTIN TRAVERS

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Synchronous Set Agreement: a Concise Guided Tour (with open problems)

Michel Raynal* Corentin Travers**

Systèmes communicants

Publication interne n ° 1791 — Mars 2006 — 20 pages

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.

Key-words: Agreement problem, Consensus, Crash failure, Distributed algorithm, Early decision, Efficiency, General omission failure, k-set agreement, Lower bound, Message-passing system, Receive omission failure, Round-based computation, Send omission failure, Synchronous system.

(Résumé : tsvp)

* IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, raynal@irisa.fr
** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, travers@irisa.fr

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

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

Une visite guidée de l’accord ensembliste synchrone

Résumé : Ce rapport constitue une visite guidé de l’accord ensembliste synchrone en preésence de crashs, de fautes d’omission en émission, et de fautes d’omission en émission et réception.

Mots clés : Systèmes répartis asynchrones, Tolérance aux fautes, Crash de processus, Accord ensembliste; Omission en émission, Omission en réception.

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 [5]. 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 $k \leq t$ [4, 15, 28]. This impossibility result generalizes the impossibility to solve the consensus problem in asynchronous systems where even only one process can crash (case $k = t = 1$) [9]. 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 [14, 19] or random numbers [20]), or restrict the input vectors that the processes can collectively propose [3, 18].

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* 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 [6]. (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 tradeoff" 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 [6]. 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., when 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* [7]. 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 [10]. 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 [16]).

Content of the paper This paper visits k-set agreement protocols in several process failure models, namely, the classical crash failure model, the send omission failure model and the more 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 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 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. (To not overload the presentation, the proofs of the protocols that are described are given in an appendix.) 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.

2 Distributed Computing Model and k-Set Agreement Problem

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 [13]. 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) [22].

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

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

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

omission failure continues to receive the messages sent by the correct process. Differently, a process that commits receive omission failures does not: it has an "autism" behavior.

2.3 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 $k > t$).

3 Set Agreement in the Crash Failure Model

3.1 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, 11, 17]). 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.)

3.2 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 [24]) 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 other processes know it has one of the k smallest values. Consequently, p_i cannot decide and stop immediately. To

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

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

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 05 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. The correctness of the protocol and that lower bound are proved in Appendix A.

3.3 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).

[1] This predicate is implicitly used in the proof of the (not-early deciding) k-set agreement protocol described in [17].

4 Set Agreement in the Send Omission Failure 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.

4.1 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 (a variant of a protocol introduced in [12]) is obtained from the basic flooding protocol by two simple modifications. They concern the lines 04 and 05.

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

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

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 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 05). 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).

4.2 Early decision and strong termination

An early-deciding k-set protocol for the send omission failure model with $t < n$, is described in [25]. 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 [25] 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 to decide.

Problem difficulty The non-early deciding protocol described in Figure 3 and the early deciding protocol described in [25] shows that the k-set agreement problem has the same lower bounds in the crash 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.

5 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.

5.1 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 [26]. 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 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 process identities associated with the processes p_ℓ that are currently trusted by p_i and that (to p_i's knowledge) trust p_j (line 06).

```
Function 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 for_each j ∈ Π do send(est_i, trusted_i) to p_j end_do end_if;
(05)    let rec_from_i = {j : (est_j, trust_j) is received from p_j during r ∧ j ∈ trusted_i};
(06)    for_each 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 (⊥) end_if;
(09)    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 ($t < \frac{n}{2}$)

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 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 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, 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 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.

A proof of this protocol can be found in [26]. 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 06-07). 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 08, and decides consequently at line 11.

5.2 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 [26]) 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 $f = k\,x + y$, with x and y being integers such that $0 < y < x$.

6 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.

6.1 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 [26]), and that protocol considers $t < n/2$. (Several protocols have been designed for the particular case $k = 1$ -consensus problem-, e.g., [21, 23, 27], 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 $t \geq n/2$ [21, 27]. It is based on a simple classical partitioning argument.

Proof In order to etablish a contradiction, let us assume that $t \geq \frac{k}{k+1}n$ and there is an algorithm $\mathcal{A}$ that solves the k-set agreement problem in synchronous systems where at most t processes can commit general omission failures. Let us partition the set of processes in $k+1$ sets $S_1, \ldots S_{k+1}$ such that $\forall 1 \leq i \leq k : |S_i| = n - t$ and $|S_{k+1}| = n - k(n-t)$. As $t \geq \frac{k}{k+1}n$, it follows that $t \geq k(n-t)$, from which we have $|S_{k+1}| = n - k(n-t) \geq n - t$. We now exhibit a run R of algorithm $\mathcal{A}$ in which $k+1$ distinct values are decided: a contradiction.

In the run R, every process that belongs to the same set S_i initially proposes the same value v_i. Moreover, the values v_i are choosen such that $i \neq j \Rightarrow v_i \neq v_j$. Processes that belong to set $S_1, \ldots, S_k$ are faulty whereas processes in set S_{k+1} are correct. The number of faulty processes is $k(n-t) \leq t$. We now describe the behavior of the faulty processes during run R. Let p_x be a faulty process that belongs to a set S_i ($i \leq k$). From the very beginning of the execution, p_x commits

- a send omission failure for each message it has to send to a process that does not belong to S_i,
- a receive omission failure each time it has to receive a message from a process that belong to set S_{k+1}.

For each $1 \leq i \leq k+1$, we construct a run R_i as follows: In run R_i, every process that does not belong to set S_i crashes before sending any message. Processes in set S_i are correct. As in run R, all processes in S_i initially propose the same value v_i. In run R_i, messages are only exchanged between processes in set S_i. Moreover, for any $p_x, p_y \in S_i$, each time algorithm $\mathcal{A}$ requires p_x to send a message to p_y, this message is delivered by p_y. As the processes in S_i are correct, it follows from the correction of algorithm $\mathcal{A}$ that they decide. Since the only value that they hear of is v_i (the only value proposed by processes in S_i), they decide that value.

Let us now consider the processes in S_i during run R. Let us first observe (O1) that a process $p_x \in S_i$ does not receive messages sent by any process that does not belong to S_i. If $i = k+1$ (i.e., p_x belongs to S_{k+1} and is a correct process), this is because a any process p_z that does not belong to S_{k+1} commits a send omission failure each time it has to send a message to p_x. In the other case ($i \neq k+1$), p_x does not receive messages from any $p_z \in S_j, j \neq i \wedge j \neq k+1$ since these processes commit send omission failures each time they have to send a message to p_x; p_x neither receive message from any $p_z \in S_{k+1}$ since it commits a receive omission failure with respect to any process that belongs to S_{k+1}.

As in run R_i, for any $p_x, p_y \in S_i$ each time algorithm $\mathcal{A}$ requires p_x to send a message to p_y, this message is delivered by p_y (O2). This is because, for any set S_i, a process that belongs to S_i does not commit omission failures with respect to the other processes of S_i. Consequently, it follows from the observations (O1) and (O2) that runs R and R_i are indistinguishable for any process that belong to S_i. This implies that in the run R, for each $1 \leq i \leq k+1$, any $p_x \in S_i$ decides v_i, from which we conclude that $k+1$ values are decided. $\square_{Theorem\ 2}$

6.2 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)
(01) est_i ← v_i; trusted_i ← Π;   % r = 0 %
(02) for r = 1, ... , t + 2 − k do
(03) begin_round
(04)   for_each j ∈ trusted_i do send est_i to p_j end_do;
(05)   foreach j ∈ trusted_i do
(06)     if (est_j received from p_j) then est_i ← min(est_i, est_j)
(07)                                  else trusted_i ← trusted_i − {j}
(08)     end_if
(09)   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 Appendix B), the design of this protocol is particularly simple. It is similar to the early-deciding uniform consensus protocol presented in [23] 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 04-09). 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 Appendix B. It is not strongly terminating.

6.3 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, 8], 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 Appendix B, for the protocol described in Figure 5.

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

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] 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, Toulouse (France), 2002.

[4] 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.

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

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

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

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

[9] 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.

[10] 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.

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

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

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

[14] 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.

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

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

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

[18] 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, Las Vegas (NV), 2005.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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)*, Changsha (China), IEEE Computer Press, pp. 23-30, 2005.

[26] 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)*, Spirnger-Verlag LNCS (to appear), Liverpool (UK), July 2006.

[27] 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.

[28] 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.

A Proof of the early-deciding protocol (crash failures)

This appendix shows that the set-agreement protocol suited to the crash failure model, that is presented in Figure 2, is correct.

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

Proof The proof of the validity consists in showing that an est_i local variable always contains a proposed variable. This is initially true (round $r = 0$). Then, a simple induction reasoning proves the property: assuming the property is true at a round $r \geq 1$, it follows from the protocol code (lines 04 and 08), and the fact that a process receives at least the value it has sent, that the property remains true at round $r + 1$. $\square_{Lemma\ 1}$

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

Proof The proof is an immediate consequence of the fact that a process executes at most $\lfloor t/k \rfloor + 1$ rounds and the computation model is the synchronous round-based computation model. $\square_{Lemma\ 2}$

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

Proof Let $EST[0]$ be the set of proposed values, and $EST[r]$ the set of est_i values of the processes that decide during r or proceed to $r + 1$ ($r \geq 1$). We first state and prove three claims.

Claim $C1$. $\forall r \geq 0$: $EST[r + 1] \subseteq EST[r]$.
Proof of the claim. The claim follows directly from the fact that, during a round, the new value of an est_i variable computed by a process is the smallest of the est_j values it has received. So values can only disappear, due to the minimum function used at line 08 or to process crashes. *End of the proof of the claim* $C1$.

Claim $C2$. Let p_i be a process such that can_decide_i is set to true at the end of r. Then est_i is one of the k smallest values in $EST[r]$.
Proof of the claim. Let v be the value of est_i at the end of r ($v \in EST[r]$). If can_decide_i is set to true at the end of r, $nb_i[r - 1] - nb_i[r] < k$ is satisfied or p_i has received a message carrying a pair $(v1, true)$, and $v1$ has been taken into account when computing the new value of est_i at line 08 during round r, i.e., $v \leq v1$. So, there is a chain of processes $j = j_a, j_{a-1}, \ldots, j_0 = i$ that has carried the boolean value $true$ to p_i. This chain is such that $a \geq 0$, $nb_j[r - a - 1] - nb_j[r - a] < k$ is satisfied, and any value v' sent by a process participating in this chain is such that $v \leq v'$ (as each process in the chain computes the minimum of the values it has received). In particular, we have $v \leq v''$ where v'' is the value sent by the first process in the chain. (The case $a = 0$ corresponds to the "one process" chain case where the local predicate is satisfied at p_i.) Due to claim $C1$, $EST[r] \subseteq EST[r - a]$. Consequently, if v'' is one of the k smallest values of $EST[r - a]$, $v \leq v''$ implies v is one of the k smallest values of $EST[r]$.

So, taking $r - a = r'$, we have to show that $nb_j[r' - 1] - nb_j[r'] < k$ implies that the value v'' of est_j at the end of r', is one of the k smallest values of $EST[r']$. As the crashes are stable, $nb_j[r' - 1] - nb_j[r'] < k$, allows concluding that p_j has received a message from all but at most $k - 1$ processes that where not crashed at the beginning of r'. As p_j computes the minimum of all the values it has received, and misses at most $k - 1$ values of $EST[r']$, this means that the value v'' computed by p_j at the end of r' is one of the k smallest values present in $EST[r']$. *End of the proof of the claim* $C2$.

Claim $C3$. Let p_i be process that decides (at line 05 or 11) during the round r. Its boolean flag can_decide_i is then equal to $true$.
Proof of the claim. The claim is trivially true if p_i decides at line 05. If p_i decides at line 11, it decides during the last round, namely $r = \lfloor t/k \rfloor + 1$. Let us consider two cases.

- At round r, p_i receives from a process p_j a message such as $can\_decide_j = true$. In that case, p_i sets can_decide_i to $true$ at line 09, and the claim follows.

- In the other case, no process p_j has decided at a round $r' < r$ (otherwise, p_i would have received from p_j a message such that $can\_decide_j = true$). Let $t = k\ x + y$ with $y < k$ (hence, $x = \lfloor t/k \rfloor = r - 1$). As $nb_i[r' - 1] - nb_i[r'] < k$ was not satisfied at each round r' such that $1 \leq r' \leq x = r - 1$, we have $nb_i[x] \leq n - kx$. Moreover, as p_i has not received from any p_j a message such that can_decide_j is equal to $true$, if, during r, p_i does not receive a message from p_j it is because p_j has crashed. So, as at most t processes crash, we have $nb_i[x + 1] \geq n - t = n - (k\ x + y)$. It follows that $nb_i[x] - nb_i[x + 1] \leq y < k$. the claim follows.

End of the proof of the claim $C3$.

To prove the lemma, we now consider two cases according to the line during which a process decides.

- No process decides at line 05. This means that a process p_i that decides, decides at line 11 during the last round. Due to the claim $C3$, such a p_i has then its flag can_decide_i equal to $true$. Due to the claim $C2$, it decides one of the k smallest values in $EST[\lfloor t/k \rfloor + 1]$.
- A process decides at line 05. Let r be the first round during which a process p_i decides at that line and v be the value it decides. Since p_i decides at r:
  - p_i has set its boolean flag can_decide_i to $true$ at the end of $r - 1$, and its estimate $est_i = v$ is consequently one of the k smallest values in $EST[r - 1]$ (Claim $C2$). It follows that two processes that decide during r decide values that are among the the k smallest values in $EST[r - 1]$.
  - p_i has sent to all the processes (line 04) the pair $(v, true)$ before deciding at line 05 during r. This implies that a (non-crashed) process p_j that does not decide at r receives v at r and uses it to compute its new value of est_j. Due to the minimum function used at line 08, it follows that, from now on, we will always have $est_j \leq v$.

    Let us assume that p_j does not crash. If it decides, it decides at $r' > r$, and then it necessarily decides a value $v' \leq v$. As $EST[r'] \subseteq EST[r - 1]$ (claim $C1$), we have $v' \in EST[r - 1]$. Combining $v' \leq v$, $v' \in EST[r - 1]$, and the fact that v is one of the k smallest values in $EST[r - 1]$, it follows that the value v' decided by p_j is one of the k smallest values in $EST[r - 1]$.

$\square_{Lemma\ 3}$

Theorem 3 [k-Set Agreement] *The protocol solves the k-set agreement problem.*

Proof The proof follows from the Lemmas 1, 2, and 3. $\square_{Theorem\ 3}$

Theorem 4 [Early Stopping] *No process halts after the round* $\min(\lfloor f/k \rfloor + 2, \lfloor t/k \rfloor + 1)$.

Proof Let us first observe that a process decides and halts at the same round; this occurs when it executes return (est_i) at line 04 or 11. As observed in Lemma 2, the fact that no process decides after $\lfloor t/k \rfloor + 1$ rounds is an immediate consequence of the code of the protocol and the round-based synchronous model. So, considering that $0 \leq f \leq t$ processes crash, we show that no process decides after the round $\lfloor f/k \rfloor + 2$. Let $f = xk + y$ (with $y < k$). This means that $x = \lfloor f/k \rfloor$.

The worst case scenario is when, for any process p_i that evaluates the local decision predicate $nb_i[r - 1] - nb_i[r] < k$, this predicate is false as many times as possible. Due to the pigeonhole principle, this occurs when exactly k processes crash during each round. This means that we have $nb_i[1] = n - k, \cdots, nb_i[x] = n - kx$ and $nb_i[x + 1] = n - f = n - (kx + y)$, from which we conclude that $r = x + 1$ is the first round such that $nb_i[r - 1] - nb_i[r] = y < k$. It follows that the processes p_i that execute the round $x + 1$ set their can_decide_i boolean to $true$. Consequently, the processes that proceed to $x + 2$ decide at line 05 during that round. As $x = \lfloor f/k \rfloor$, they decide at round $\lfloor f/k \rfloor + 2$. $\square_{Theorem\ 4}$

B Proof of the protocol for general omission failures when $t < \frac{k}{k+1}n$

This appendix shows that the set-agreement protocol for the general omission failure model, that is presented in Figure 5, is correct. The proof uses the following notations :

- $\mathcal{C}$ 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").
- $Completing_i[r] = \{i : p_i$ proceeds to $r+1\}$. By definition, $Completing_i[0] = \Pi$. If $r = t-k+2$, "p_i proceeds to $r+1$" means p_i executes line 12.
- $EST[r] = \{est_i[r] : i \in Completing[r]\}$. By definition $EST[0]$ = the set of proposed values.
  $EST[r]$ contains the values that are present in the system at the end of round r.
- For any relation $\mathcal{R} \in \{=, >, \geq, <, \leq\}$, $P_{\mathcal{R}}(v, r) = \{i : (i \in Completing[r]) \wedge (est_i[r]\ \mathcal{R}\ v)\}$. As an example, $P_=(v, r)$ is the set of processes p_i that proceed to $r+1$ with an estimate equal to v at the end of round r (i.e., such that $est_i[r] = v$).

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

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

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. $\square_{Lemma\ 4}$

Lemma 5 $|EST[1]| \leq t+1$.

Proof Let $v_{min} = \min\{est_i[0], i \in \mathcal{C}\}$. The proof uses the following sets of processes/values:
$\mathcal{F}_- = \{i \in \Pi - \mathcal{C} : est_i[0] < v_{min}\}$,
$\mathcal{F}_+ = \{i \in \Pi - \mathcal{C} : est_i[0] > v_{min}\}$,
$V_- = \{v : \exists i \in \mathcal{F}_-$ such that $v = est_i[0]\}$,
$V_+ = \{v : \exists i \in \mathcal{F}_+$ such that $v = est_i[0]\}$.
$\mathcal{F}_-$ (resp., $\mathcal{F}_+$) is the set of faulty processes that propose a value strictly smaller (resp., strictly greater) than v_{min}. V_- (resp. V_+) is the set of values strictly smaller (resp., strictly greater) than v_{min} that are proposed by faulty processes.

Let p_i be a correct process. During the first round, it receives and processes all the values proposed by the correct processes. As a process updates its estimate by taking the smallest value it has received (line 06), we have $est_i[1] \leq v_{min}$. It follows that p_i can update its estimate only to v_{min} or a value sent by a process $p_j : j \in \mathcal{F}_-$, i.e., $\{est_i[1] : i \in \mathcal{C}\} \subseteq \{v_{min}\} \cup V_-$ (O1).

Let $i \in \mathcal{F}_-$. As $est_i[0] < v_{min}$, p_i can update its estimate only to a value received from a process that belong to $\mathcal{F}_-$. This implies that $\{est_i[1] : i \in \mathcal{F}_-\} \subseteq V_-$ (O2).

Let $i \in \mathcal{F}_+$. In that case, p_i can adopt a value from any process in Π. It follows that, at the end of the first round, there are at most $|\mathcal{F}_+|$ distinct values among the processes that belong to $\mathcal{F}_+$ (O3).

From (O1) and (O2), we obtain: $\{est_i[1] : i \in \mathcal{C} \cup F_-\} \subseteq \{v_{min}\} \cup V_-$. Moreover, we have $|V_-| \leq |F_-|$ (O4). From (O3), (O4) and the fact that $|F_-| + |\mathcal{F}_+| \leq t$, we conclude that at the end of the first round we have $|\{est_i[1] : i \in Completing[1]\}| \leq t+1$. $\square_{Lemma\ 5}$

Lemma 6 $\forall r, 1 \leq r \leq t-k+2$, *we have* $|EST[r]| \leq t-r+2$.

Proof The proof is by induction on the round number r. For all r, $1 \leq r \leq t-k+2$, let $HR(r)$ be the predicate $|EST[r]| \leq t-r+2$. The base case of the induction (i.e., $HR(1)$) follows directly from Lemma 5. So, considering the induction assumption (namely, $HR(x)$ is satisfied for all x such that $1 \leq x < r$), we prove that $HR(r)$ is satisfied. Let us first observe that if $|EST[r-1]| < t-(r-1)+2$, $HR(r)$ follows directly from Lemma 4 as $|EST[r]| \leq |EST[r-1]| \leq t-r+2$. So, the rest of the proof assumes $|EST[r-1]| = t-(r-1)+2 = t-r+3$.

In the following, we consider a particular value $v_m \in EST[r-1]$ and the set of processes that have an estimate equal to v_m at the end of the round $r-1$. The value v_m is defined as follows:

$$v_m = \max\{v : |P_=(v, r-1)| < n-t\}$$

(among the values of $EST[r-1]$, v_m is the greatest one that is the estimate of less than $n-t$ processes that complete the round $r-1$).

We claim that v_m exists (*Claim C1*) and $v_m \notin EST[r]$ (*Claim C2*). Assuming these claims, it follows from $v_m \in EST[r-1]$, and $|EST[r-1]| = t-r+3$ (Case assumption), that $|EST[r]| = t-r+2$, which proves $HR(r)$, from which the Lemma follows. The rest of the proof concerns the claims *C1* and *C2*.

Claim C1: $\exists v$ such that $|P_=(v, r-1)| < n-t$.
Proof of the Claim C1. As $r \leq t-k+2$ (Lemma assumption), we have $t-r+2 \geq k$. Combining this with $|EST[r-1]| = t-r+3$ (Case assumption), we obtain $|EST[r-1]| > k$.

We now show the existence of v by contradiction. Let us suppose that each value $v' \in EST[r-1]$ is such that $|P_=(v', r-1)| \geq n-t$ (i.e., there are at least $n-t$ copies of each value present in the system at the end of round $r-1$). As $|EST[r-1]| > k$ and at most n processes complete round $r-1$, the inequality $n \geq (k+1)(n-t)$ must hold. Thus, $(k+1)t \geq (k+1)n - n$ from which we obtain that $t \geq \frac{kn}{k+1}$. This contradicts the upper bound on t, namely, $\frac{kn}{k+1} > t$. *End of the Proof of the Claim C1.*

Claim C2: $v_m \notin EST[r]$.
Proof of the Claim C2. If $est_i[0] = v$, we say "p_i learns v during the (fictitious) round 0". More generally, for $d \geq 1$, we say "p_i learns v during round d" if p_i (1) completes the round d, (2) has never received v by the end of round $d-1$ and, (3) has an estimate equal to v at the end of round d. This means that (1) $i \in Completing[d]$, (2) $\forall d' < d : est_i[d'] > v$ and, (3) $est_i[d] = v$. We consider two cases: no process learns v_m during round $r-1$ (case 1); a process learns v_m during round $r-1$ (case 2).

Case 1: No process learns v_m during $r-1$.
We claim (*Sub-claim C2.1*) $est_i[r] = v_m \Rightarrow |trusted_i[r]| < n-t$. The proof that $v_m \notin EST[r]$ follows directly from this claim as then a process p_i that sets its estimate to v_m after having executed the lines 05-09 during round r necessarily returns $\perp$ at line 10. This implies that any process p_i that completes round r is such that $est_i[r] \neq v_m$, i.e., $v_m \notin EST[r]$.

The proof of *C2.1* is based on the following properties (implicitly defined in the context of the case assumption):
Property *P1*: $\forall i \in P_>(v_m, r-1) : P_=(v_m, r-1) \subseteq \Pi - trusted_i[r-1]$.
This property states that any process p_x completing the round $r-1$ with an estimate value equal to v_m, is not trusted by any process p_i that completes the round $r-1$ with an estimate value greater than v_m.
Property *P2*: $(x \in Completing[r]) \wedge (est_x[r] = v_m) \Rightarrow x \in P_=(v_m, r-1)$.
This property states that any process p_x completing the round r with an estimate value equal to v_m, was such that $est_x[r-1] = v_m$.
We prove first *P1* and *P2*, and then *C2.1*.

Property P1: $\forall i \in P_>(v_m, r-1) : P_=(v_m, r-1) \subseteq \Pi - trusted_i[r-1]$.
Proof of P1. Let $i \in P_>(v_m, r-1)$ and $x \in P_=(v_m, r-1)$. Let us first observe that it follows from the fact that no process learns v_m during round $r-1$ that $P_=(v_m, r-1) \subseteq P_=(v_m, r-2)$. Consequently, $x \in P_=(v_m, r-2)$ and thus a message sent by p_x during round $r-1$ carries $v_m (= est_x[r-2])$ (line 04).

As $est_i[r-1] > v_m$ and p_x sends v_m during round $r-1$, it follows from the $\min()$ function used to compute new estimates (line 06) that either p_i does not receive the message from p_x during round $r-1$ or p_i does not trust p_x at the

beginning of round $r-1$. In both cases, p_i does not trust p_x at the end of round $r-1$, i.e., $x \in \Pi - trusted_i[r-1]$. *End of the Proof of P1*

Property P2: $(x \in Completing[r]) \wedge (est_x[r] = v_m) \Rightarrow x \in P_=(v_m, r-1)$.
Proof of P2. Let $x \in Completing[r]$ such that $est_x[r] = v_m$. Due to the $\min()$ function used by a process to update its estimate (line 06), $x \notin P_<(v_m, r-1)$ (1).

Consider now a process p_i that belongs to $P_>(v_m, r-1)$. Let us observe that only the processes in $P_=(v_m, r-1)$ can send v_m during the round r. Due to property *P1*, when p_i completes the round $r-1$, it does not trust the processes belonging to $P_=(v_m, r-1)$. Consequently, p_i does not consider the messages sent during the round r by the processes in $P_=(v_m, r-1)$. It follows that $est_i[r] \neq v_m$, from which we conclude that $x \notin P_>(v_m, r-1)$ (2).

As the sets $P_<(v_m, r-1)$, $P_=(v_m, r-1)$ and $P_>(v_m, r-1)$ define a partition of $Completing[r-1]$, and $x \in Completing[r] \subseteq Completing[r-1]$, it follows from (1) and (2) that $x \in P_=(v_m, r-1)$. *End of the Proof of P2.*

Claim C2.1: $est_x[r] = v_m \Rightarrow |trusted_x[r]| < n-t$.
Proof of the Claim C2.1. Let $x \in Completing[r]$ such that $est_x[r] = v_m$. Let us first consider a process p_i that belongs to $P_<(v_m, r-1)$. If p_i sends messages during round r, these messages necessarily carry a value $< v_m$ (line 04). Since $est_x[r] = v_m$, either p_x does not receive this message from p_i (lines 06-08) or p_x does not trust p_i at the beginning of round r. In both cases, we have $i \notin trusted_x[r]$, from which we conclude that $P_<(v_m, r-1) \subseteq \Pi - trusted_x[r]$ (1).

Let now p_i be a process that belongs to $P_>(v_m, r-1)$. Due to property *P2*, $x \in P_=(v_m, r-1)$. Consequently, $x \notin trusted_i[r-1]$ (property *P1*). This implies that p_i does not send a message to p_x during round r (line 04). Thus, $i \notin trusted_x[r]$. Since this holds for any process that belongs to $P_>(v_m, r-1)$, we have $P_>(v_m, r-1) \subseteq \Pi - trusted_x[r]$ (2).

By combining (1) et (2), we obtain $trusted_x[r] \subseteq \Pi - (P_<(v_m, r-1) \cup P_>(v_m, r-1))$. Moreover, as only processes that belong to $Completing[r-1]$ may send messages during round r, it follows from the fact that the sets $P_<(v_m, r-1), P_=(v_m, r-1)$ and $P_>(v_m, r-1)$ define a partition of $Completing[r-1]$ that $trusted_x[r] \subseteq P_=(v_m, r-1)$. Finally, due to the definition of v_m, we have $|P_=(v_m, r-1)| < n-t$ from which we conclude that $trusted_x[r]| < n-t$. *End of the Proof of the Claim C2.1.*
End of the Proof of Case 1 of the Claim C2.

Case 2: There is a process that learns v_m during round $r-1$.
We claim (*Claim C2.2*), in the case assumptions, $v_m = \max(EST[r-1])$. We prove by contradiction that $v_m \notin EST[r]$. Let us assume that there is a process p_x such that $x \in Completing[r] \wedge est_x[r] = v_m$. Due to the $\min()$ function used by p_x to update its estimate (line 06), p_x receives during round r from the processes it trusts only values $\geq v_m$. As (1) only processes that belong to $P_\geq(v_m, r-1)$ can send values $\geq v_m$ during round r and (2) $P_\geq(v_m, r-1) = P_=(v_m, r-1)$ (Claim *C2.2*), p_x can only trust processes that belong to $P_=(v_m, r-1)$. Moreover, due to the very definition of v_m, $|P_=(v_m, r-1)| < n-t$. This implies that $|trusted_x[r]| < n-t$ from which we conclude that p_x returns $\perp$ at line 12: a contradiction with $x \in Completing[r]$.

Claim C2.2: $v_m = \max(EST[r-1])$. (Let us remind that, in the context of this claim, $|EST[r-1]| = t-r+3$ and at least one process learns v_m during $r-1$.)
Proof of the Claim C2.2. Let $\alpha = |\{v : v \in EST[r-1] \wedge v < v_m\}|$ and, $\beta = |\{v : v \in EST[r-1] \wedge v > v_m\}|$. To prove that v_m is the greatest value in $EST[r-1]$, the rest of the proof establishes $\beta = 0$. Let us notice that $|EST[r-1]| = \alpha + \beta + 1 = t-r+3$.

Let us define three sets of processes, denoted A, B and C, as follows:

- Let $A = \{i : est_i[0] < v_m\}$. A is the set of processes that propose a value strictly smaller than v_m. Since values can only disappear while rounds progress (Lemma 4), clearly $|A| \geq \alpha$.
- Let $B = \{i : est_i[r-1] > v_m \wedge i \in Completing[r-1]\}$ (or $B = \bigcup_{v \in EST[r-1] \wedge v > v_m} P_=(v, r-1)$). B is the set of processes that have an estimate strictly greater than v_m at the end of round $r-1$. Due to the very definition of v_m, for each $v \in EST[r-1]$ such that $v > v_m$, we have $|P_=(v, r-1)| \geq n-t$. It follows that $|B| \geq \beta(n-t)$.

- Let p_x be a process that learns v_m during round $r-1$. We claim (*Claim C2.3*) that there is a chain of r distinct processes $p_{v_m(0)}, \ldots, p_{v_m(r-1)} (= p_x)$ such that $\forall \ell : 1 \leq \ell \leq r-1 : p_{v_m(\ell)}$ learns v_m at round ℓ from $p_{v_m(\ell-1)}$ (hence, $est_{v_m(\ell)}[\ell] = v_m$). The set C is the set of processes that participate in the chain that carries v_m to p_x. More precisely, $C = \{v(0), \ldots, v(r-1) = x\}$. We clearly have $|C| = r$.

We now show that any pair of sets A, B and C has an empty intersection.

- $A \cap B = \emptyset$ and $A \cap C = \emptyset$.
  Let $a \in A$ ($est_a[0] < v_m$) and $x \in B \cup C$. There is a round $d \leq r-1$ such that $est_x[d] \geq v_m$ (O1). (This follows from the following observations. If $x \in B$, taking $d = r-1$ establishes (O1). If $x \in C$, (O1) follows from the definition of the set C,namely, p_x learns v_m during some round $d \leq r-1$, at the end of which $est_x[d] = v_m$.)

  It follows from (O1) and the relation $\forall\ \ell, \ell' : 0 \leq \ell < \ell' \leq r-1 \ \Rightarrow est_x[\ell] \geq est_x[\ell']$ that $est_x[0] \geq v_m$. Consequently, as $est_a[0] < v_m$ (definition A), we have $A \cap B = \emptyset$ and $A \cap C = \emptyset$.

- $B \cap C = \emptyset$.
  Let p_x be a process that belongs to C. If $x \notin Completing[r-1]$, $x \notin B$ (a process that belongs to B completes round $r-1$). If $x \in Completing[r-1]$, as there is a round ℓ, $1 \leq \ell \leq r-1$, such that $est_x[\ell] = v_m$, and due to the $\min()$ function an estimate value can only decrease, we have $est_x[r-1] \leq v_m$. Consequently, as every process p_y that belongs to B is such that $est_y[r-1] > v_m$, we have $x \notin B$, from which we conclude that $B \cap C = \emptyset$.

We have established the following relations:

- (1) $A \cap B = \emptyset$, $A \cap C = \emptyset$ and $B \cap C = \emptyset$,
- (2) $|A| \geq \alpha$, $|B| \geq \beta(n-t)$ and $|C| = r$,
- (3) $\alpha + \beta + 1 = t - r + 3$.

By combining (1) and (2) we obtain: $n \geq \alpha + \beta(n-t) + r$, i.e., $n - \beta(n-t) - r \geq \alpha$. The last inequality combined with (3) gives $n - \beta(n-t) - r \geq t - r + 3 - \beta - 1$ from which we have: $n - t - 1 > \beta(n-t-1)$. This implies $\beta = 0$[2]. Since β is the number of values greater than v_m in $EST[r-1]$, it follows that v_m is the greatest value that belongs to $EST[r-1]$. *End of Proof of Claim C2.2.*

Claim C2.3: Let p_i be a process that learns a value v at round $d \geq 1$. There is a chain of $d+1$ distinct processes $p_{v(0)}, \ldots, p_{v(d)} (= p_i)$ such that $\forall\ \ell, 1 \leq \ell \leq d : p_{v(\ell)}$ learns v at round ℓ from $p_{v(\ell-1)}$.
Proof of Claim C2.3. We prove the claim by induction on d.

[Base case: $d = 1$] Suppose that a process p_i learns a value v during the first round. Then, a process $p_{v(0)}$ has sent v to p_i during that round. It follows that there are two distinct processes p_i and $p_{v(0)}$ such that $p_{v(0)}$ learned v during the round 0 and and p_i learns it during the first round.

[Induction case: $d > 1$] Suppose the claim holds for $1 \leq d' < d$. Let p_i be a process that learns a value v at round d from some process p_x. During the round d, p_i can process messages only from processes it has never suspected from the first round until the round $d-1$ (included). It follows that p_i has received a message from p_x at each round $1, \ldots, d-1$. Since a process that learns a value during a round forwards that value during the next round, we conclude that p_x learnt v at round $d-1$. By applying the induction hypothesis to p_x, we conclude that there is a chain of $d+1$ distinct processes that participated in forwarding the value v to p_i (that learns it at round d). *End of the Proof of the Claim C2.3.*
End of the Proof of Case 2 of the Claim C2.

$\square_{Lemma\ 6}$

Theorem 5 [Agreement] *No more than k different values are decided.*

[2] Notice that $n - t - 1 \neq 0$. Let us remind that the protocol assumes $k \leq t < \frac{kn}{k+1}$ (if $k > t$ a one round protocol solves trivially the problem). Suppose $t = n-1$. As $t < \frac{kn}{k+1}$, we then have $n - 1 < \frac{kn}{k+1}$, which implies $n - 1 < k$. On the other side $k \leq t$, i.e., $k \leq n-1$. A contradiction.

Proof Let us first notice that a decided value belongs to the set $EST[t-k+2]$. Due to Lemma 6, $|EST[t-k+2]| \leq t-(t-k+2)+2 = k$. Consequently, at most k distinct values are decided. $\square_{Theorem\ 5}$

Theorem 6 [Termination] *Every correct process decides.*

Proof

Let us first observe that it follows from the protocol text that a process executes at most $t-k+2$ rounds (line 02). A process that does not crash nor returns $\perp$ at line 08 decides when it executes the return statement of line 12. As a correct process does not crash, we have to show that a correct process never returns $\perp$ at line 08. More precisely, we prove by induction on the round number the following property: $\forall i \in \mathcal{C}, \forall r, 1 \leq r \leq t-k+2$: (1) $\mathcal{C} \subseteq trusted_i[r]$ and, (2) $\mathcal{C} \subseteq Completing[r]$. Let p_i be a correct process.

- [Base case] Let us consider any correct process p_j. Let us first observe that we have initially $trusted_j[0] = \Pi$ (line 01). It follows then from line 04 that p_j sends a message to p_i during the first round. As both p_i and p_j are correct and $j \in trusted_i[0]$, this message is received and processed (at line 06) by p_i during the first round. Consequently, p_i does not remove j from $trusted_i$. Since this is true for any correct process p_j, $\mathcal{C} \subseteq trusted_i[1]$ which proves item (1). As $|\mathcal{C}| \geq n-t$, p_i does not return $\perp$ at line 08 during the first round, i.e., $i \in Completing[1]$ which proves item (2).

- [Induction case] Let us assume that properties (1) and (2) hold from the first round until round $r-1 (r \geq 2)$ for any correct process. First of all, let us notice that any correct process p_j sends a message to p_i during round r. This follows from the induction assumption: as $j \in Completing[r-1]$ and $\mathcal{C} \subseteq trusted_j[r-1]$, p_j send a message to p_i at line 04. Moreover, as both p_i and p_j are correct and $j \in \mathcal{C} \subseteq trusted_i[r-1]$, this message is received and processed by p_i during round r. The proof is now the same as the base step, replacing $trusted_i[1]$ by $trusted_i[r]$, and $Completing[1]$ by $Completing[r]$.

$\square_{Theorem\ 6}$

Theorem 7 [Validity] *A decided value is a proposed value.*

Proof Let us first observe that a process p_i decides the value $est_i[t-k+2]$. This means that the set of decided value is a subset of $EST[t-k+2]$. Due to Lemma 4, $EST[t-k+2] \subseteq EST[0]$, which is the set of proposed values. $\square_{Theorem\ 7}$