ABSTRACT

The \textit{k-Set Agreement} problem generalizes the consensus problem (which corresponds to the case \( k = 1 \)). The processes propose values and each correct process has to decide a value such that (1) a decided value is a proposed value, and (2) no more than \( k \) distinct values are decided. Let \( f \) be the maximum number of processes that can crash. It has first been shown that the consensus problem cannot be solved in asynchronous distributed systems when \( f > 0 \) (this is the well-known FLP’s impossibility result). It has then been shown that this impossibility still holds for the \( k \)-set agreement problem when \( f \geq k \).

In the case of the consensus problem, two main approaches have been investigated to circumvent this impossibility: randomization and unreliable failure detectors. For the more general case of the \( k \)-set agreement problem, the failure detector approach has recently been investigated.

This paper presents a randomization approach to solve the \( k \)-set agreement problem in asynchronous distributed systems in which \( f \geq k \). It requires that at least a majority of processes be correct. The proposed protocol does not require the a priori knowledge of the set of values proposed by processes. It relies on a relatively simple combination of reliable broadcast with randomization. Interestingly, the proposed protocol shows that more choices allow more faults or more efficient runs.

Keywords
Asynchronous system, Consensus, Fault-tolerance, \( k \)-Set Agreement, Randomized Protocol.

1. INTRODUCTION

Agreement problems are crucial when designing and implementing fault-tolerant applications on top of an asynchronous distributed system prone to failures. Among the agreement problems, Consensus has gained a leadership position as it can be seen as their “greatest common subproblem”. That is why this problem is considered as a fundamental distributed computing problem. Informally, the consensus problem can be defined in the following way. Each process proposes a value, and each correct process has to decide a value such that (1) there is a single decided value, and (2) the decided value is a proposed value.

Solving the consensus problem in asynchronous distributed systems where processes may crash is far from being trivial. More precisely, it has been shown by Fischer, Lynch and Paterson (FLP result [7, 10]) that there is no solution to the consensus problem in those systems as soon as processes (even only one) may crash. So, an asynchronous distributed system where processes can crash requires additional assumptions to solve the problem. Two main approaches have been investigated. Both are oracle-based. The first one consists in equipping each process with a random number generator [2, 16, 17]. The random oracles allow the correct processes to eventually decide with probability 1 (\(^1\)). The second major approach to circumvent this impossibility result lies in the Unreliable Failure Detector concept [4]. In that case, each process is equipped with an oracle that provides it with a list of processes suspected to have crashed. Several types of failure detectors can be defined [4, 5], and associated protocols have been designed [4, 13]. The approach that consists in combining failure detection and randomization has also been investigated [1, 15].

Recently, a new Condition-based approach has been proposed to solve consensus [11]. It consists in identifying sets of input vectors for which it is possible to design a consensus protocol that works despite up to \( f \) faults. Such conditions actually define a strict hierarchy [12].

The \textit{k-Set Agreement} problem generalizes the consensus problem: it allows up to \( k \) different values to be decided. This problem has been introduced by S. Chaudhuri [6]. Her motivation was to study whether the number of choices allowed to processes (namely, \( k \)) is related to the maximum number of processes that can crash (let \( f \) be this number). While there are trivial solutions when \( f < k \), it has been shown that the \( k \)-set agreement problem has no solution when \( f \geq k \) [3, 9, 19]. To circumvent this impossibility result, similarly to the consensus case, the failure detector approach has recently been investigated. Particular failure detectors are used in [20]. A more general Limited Accuracy Failure Detector concept has been introduced in [14] to solve the \( k \)-set agreement problem.

\(^1\)It is also possible to view the randomized approach as solving a weaker problem, because a probability is attached to the termination property.
This paper investigates the randomization approach to solve the \( k \)-set agreement problem. It proposes a protocol that combines randomization and reliable broadcast. Basically, randomization is used to allow the processes to converge, while reliable broadcast is used to allow them to disseminate the values they propose. The protocol requires that at least a majority of processes be correct. It does not require a priori knowledge of the set of values that can be proposed by the processes. Additionally, the protocol exhibits an interesting tradeoff relating the maximum number of processes it allows to crash with its expected convergence speed: more choices allow either more faults or more efficient runs. Last but not least, to our knowledge, this is the first \( k \)-set agreement randomized protocol.

The paper is made up of six sections. Section 2 introduces the computation model and defines the \( k \)-set agreement problem. Section 3 presents the protocol, while Section 4 proves it correct. Then, Section 5 discusses the protocol (cost, improvement, assumptions on \( f \)). Finally, Section 6 concludes the paper.

2. DISTRIBUTED SYSTEMS, RANDOM ORACLES AND \( k \)-SET AGREEMENT

2.1 Asynchronous Distributed Systems with Process Crash Failures

The computation model follows the one described in [4, 7]. We consider a system consisting of a finite set \( \Pi \) of \( n > 1 \) processes, namely, \( \Pi = \{p_1, \ldots, p_n\} \). A process can fail by crashing, i.e., by prematurely halting. It behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition, a correct process is a process that does not crash. A faulty process is one that is not correct. Let \( f \) denote the maximum number of processes that may crash. (Conditions on \( f \) are discussed in Sections 5.3 and 5.4.)

Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are not required to be FIFO, but are assumed to be reliable: they do not create, duplicate, alter or lose messages. There is no assumption on the relative speed of processes nor on message transfer delays (i.e., the system is asynchronous [10]).

2.2 Random Oracles

A random oracle consists of a set of R-oracle modules, each attached to a process [16]. The R-oracle module attached to \( p_i \) provides it with a value \( x \in \{1, \ldots, n\} \) each time \( p_i \) invokes the primitive \( \text{random} \). A uniform distribution is assumed; this means that each value \( x \) (\( 1 \leq x \leq n \)) has probability \( 1/n \) to be returned when \( p_i \) invokes \( \text{random} \).

2.3 The \( k \)-Set Agreement Problem

The \( k \)-set agreement problem [6] has been informally stated in the Introduction: each process proposes a value, and all correct processes have to decide on a value such that (1) any decided value has been proposed, and (2) no more than \( k \) different values are decided. In the context of asynchronous distributed systems augmented with random oracles, the \( k \)-set agreement problem is formally defined by the following properties:

- **Termination:** With probability 1, every correct process eventually decides a value.
- **Validity:** If a process decides \( v \), then \( v \) was proposed by some process.
- **Uniform Agreement:** At most \( k \) different values are decided by processes.

Moreover, we assume that the set of values that can be proposed by the processes is not known in advance.

3. A RANDOMIZED \( k \)-SET AGREEMENT PROTOCOL

3.1 Underlying Principles

To solve the \( k \)-set agreement problem, the proposed protocol combines \text{Reliable Broadcast} and randomization. The processes use reliable broadcast to disseminate the values they propose. Randomization is used to ensure termination with probability 1. Asynchronous rounds are used to make the processes converge. And, last but not least, a default value (\( \perp \)) is used during each round to restrict the set of proposed values to be a set of at most \( k \) values.

3.2 Reliable Broadcast

\text{Reliable Broadcast} is made up of two communication primitives: \( R_{\text{Broadcast}}(m) \) and \( R_{\text{Deliver}}(m) \). When a process issues \( R_{\text{Broadcast}}(m) \), we say that it “\( R \)-broadcasts” \( m \). Similarly, when a process issues \( R_{\text{Deliver}}(m) \), we say that it “\( R \)-delivers” \( m \). Reliable broadcast is defined by the following set of properties [8]:

- **Termination:** If a correct process \( R \)-broadcasts \( m \), then any correct process \( R \)-delivers \( m \) (no message from a correct process is lost).
- **Uniform Agreement:** If a process \( R \)-delivers \( m \), then any correct process \( R \)-delivers \( m \) (no message \( R \)-delivered by a -correct or not- process is missed by a correct process).
- **Validity:** If a process \( R \)-delivers \( m \), then \( m \) has been \( R \)-broadcast by some process (no spurious message).
- **Integrity:** A process \( R \)-delivers a message \( m \) at most once (no duplication).

Implementations of reliable broadcast can easily be designed for asynchronous systems. A very simple (but inefficient) one, that works in fully connected networks, is the following: when a process receives a message \( m \) for the first time, it first forwards \( m \) to all the other processes, and only then considers the delivery of \( m \) [8]. According to the underlying network topology and the way message identifiers are built, more efficient implementations can be designed [18].

3.3 Structure of The Protocol

The protocol is described in Figure 1. Each process \( p_i \) starts its participation in the protocol by invoking the function \( k\text{-Set Agreement}(v_i) \) which returns a decided value. A process \( p_i \) obtains its decision value \( v \) when it invokes return (line 16 or 4). The execution of this invocation terminates the participation of \( p_i \) in the protocol.
To prevent a process from blocking forever (i.e., waiting for a value from a process that has already been decided), a process that decides uses again a reliable broadcast (lines 4 and 12) to disseminate its decision value. Hence, for a process \( p_i \), the protocol is made up of two tasks. The task \( T1 \) handles the messages that are R-delivered. The task \( T2 \) constitutes the core of the protocol.

### 3.4 Core of the Protocol

Each process manages a local variable (\( esti \)) that represents the current estimate of its decision value. Initially, \( esti \) is set to \( v_i \) (the value proposed by \( p_i \)). When the initial value of \( p_i \) is R-delivered by \( p_i \), it is saved in \( val[j] \). Initially, \( val[j] \) is set to \( \bot \).

The processes execute a sequence of asynchronous rounds to converge to a set of at most \( k \) values. Each round is made up of two communication phases (hence it costs two communication steps). The aim of the first phase is to force each process to adopt either a value from a set of at most \( k \) values plus possibly \( \bot \). The adopted value is kept in \( ph2est \) (line 10).

**First phase of a round.** The processes first exchange their current estimate values (lines 7-8). Let us note that, as far as the round \( r \) is concerned, a PHASE1 \((r,v)\) message can be interpreted as a vote for the value \( v \). Accordingly, a process \( p_i \) adopts a value if it has received enough votes for it, say \( W \) votes. If, among the values it has received, none has enough votes to be adopted, \( p_i \) adopts the default value (\( \bot \)). The adopted value is kept in \( ph2est \) (line 10).

The aim is to have no more than \( k \) values adopted by the processes at the end of the first phase. In order to attain this aim, we must have \( (k+1)W > n \) (as there are only \( n \) processes, \( k+1 \) values cannot each obtain \( W \) votes). This means that \( W = \lceil (n+1)/(k+1) \rceil = \lceil n/(k+1) \rceil + 1 \).

Let us now examine how many PHASE1 \((r,v)\) messages a process has to wait for (at line 8) before adopting a value (at line 10) in order it gets a chance to adopt a value initially proposed by a process (i.e., a value different from \( \bot \)). Let \( R \) be this number. Considering the case where \( p_i \) adopts a non-\( \bot \) value, let us examine the worst situation: \( p_i \) can receive \((W-1)\) votes for \((k-1)\) different values, and only then receive \( W \) votes for the value \( v \) it adopts. Hence, \( R = (W-1)(k-1) + W = (W-1)k + 1 \). Moreover, in order that no process blocks at line 8, we must have \( R \leq (n-f) \) (for the constraint on \( f \), see the discussion in Section 5.4).

Hence, at the end of the first phase, the set of \( ph2est \), local variables contains at most \( k \) values, plus possibly \( \bot \). The aim of the second phase is to allow each process to decide one of these non-\( \bot \) values in such a way that the Agreement property be not violated if processes decide during different rounds.

**Second phase of a round.** During the second phase, the processes exchange the values they have previously adopted. A process \( p_i \) waits for \( ph2est \) messages from a majority of processes (lines 12-13). As shown at line 13, \( ph2reci \) denotes the set of values received by \( p_i \). Let us notice that if \( v (\neq \bot) \) belongs to \( ph2reci \), then \( v \) was the estimate of...
at most $W$ processes at the beginning of the current round. There are three cases determined by the content of $ph2_{rec}$:

- If $⊥ ∉ ph2_{rec}$, $p_i$ decides on any value of this set (lines 15-16).
- If $ph2_{rec}$ contains both $⊥$ and non-$⊥$ values, $p_i$ updates its estimate (est$_i$) to any of those non-$⊥$ values, and proceeds to the next round.
- If $ph2_{rec}$ contains only $⊥$, $p_i$ updates its current estimates (est$_i$) to a randomly chosen value (line 14), and then proceeds to the next round. Actually, $p_i$ randomly selects a process identity (say $x$) and sets est$_i$ to val$_i[x]$. Let us note that val$_i[x]$ is equal to the value proposed by $p_x$ or $⊥$. The randomness of the choices guarantees that eventually there are rounds during which $p_i$ selects non-$⊥$ entries of its val$_i$ array.

4. PROOF

The proof of the Validity property is left to the reader (hint: note that a decided value is different from $⊥$, and any estimate variable est$_i$ can only contain a proposed value or $⊥$). As announced previously, the proof of the Termination property assumes $R \leq (n - f)$, i.e., $f < n - k[n/(k + 1)]$ (see the discussion in Section 5.4).

4.1 Preliminary Lemmas

**Lemma 4.1.** If no process decides during round $r' \leq r$, then all correct processes will start the round $r + 1$.

**Proof.** The proof is by contradiction. Let $r$ be the first round during which a correct process blocks forever. It does it at line 8 or 12 (wait statement).

By assumption all the correct processes start the round $r$, and consequently send a phase1($r$, $-$) message. As there are at least $(n - f)$ correct processes, and $(n-f) \geq R$, it follows that any correct process $p_i$ receives at least $R$ phase1($r$, $-$) messages. Hence $p_i$ cannot block forever at line 8. Additionally, it follows that all the correct processes send a phase2($r$, $-$) message at line 11.

As $∀k$: $((n - k[n/(k + 1)]) \leq n/2)$, and $R = k[n/(k + 1)] + 1$, it follows that $((n - f) \geq R > n/2)$. Hence, every correct process $p_i$ receives a phase2($r$, $-$) from a majority of processes. Consequently, $p_i$ cannot block forever at line 12. □

**Lemma 4.2.** Let $EST(r)$ be the set of the estimate values of the processes that start a round $r$. If $⊥ ∉ EST(r)$ and $|EST(r)| \leq k$, then any process that starts $r$ decides during $r$ unless it crashes.

**Proof.** As by assumption there are no more than $k$ different estimates values (all different from $⊥$) at the beginning of $r$, the set of the phase1($r$, $-$) messages carry no more than $k$ different values. As $R = (W - 1)k + 1$, it follows that any process $p_i$ (that executes line 9 during $r$) selects one of these values (say $v$) to update $ph2_{est}$, at line 10. Said in another way, no $ph2_{est}$ local variable is set to $⊥$. It follows that $∀i$: $⊥ ∉ ph2_{rec}$. Consequently, any process $p_i$ can only execute line 15, and accordingly decide. □

**Lemma 4.3.** Let $PH2_{EST}(r)$ be the set including the values of all the $ph2_{est}$ local variables at the end of the first phase of $r$ (i.e., just after line 10). $PH2_{EST}(r)$ contains at most $k$ values, plus possibly the default value $⊥$.

**Proof.** Let us assume that $PH2_{EST}(r)$ contains $k + 1$ non-$⊥$ values. If a value belongs to this set (it is the value of a $ph2_{est}$ local variable), it has been received (by some $p_i$) from at least $W$ processes (see line 9). Moreover, each process sends a single phase1 message carrying a single value (line 7). It follows that $(k + 1)W$ processes have sent phase1 messages. As $(k + 1)W > n$, this is impossible. □

4.2 Termination

**Theorem 4.4.** Every correct process eventually decides with probability 1.

**Proof.** Let us remark that if a process decides then all correct processes decide: this is due to the reliable broadcast primitive used to disseminate a decided value (lines 16 and 4). The proof is by contradiction. Let us assume that no process decides. There is a time $t$ after which:

- (H1) There are only correct processes executing the protocol, and
- (H2) The val arrays of the correct processes are equal. This is because these arrays are filled in with values that are disseminated with a reliable broadcast primitive. If both $p_i$ and $p_j$ are correct, then if the value $v_k$ is $R$ delivered by $p_i$, it is also $R$ delivered by $p_j$. Hence after $t$, val$_i[k] = v_k$ implies val$_j[k] = v_k$.

Let us first note that, as no process decides, no correct process blocks forever in a round (Lemma 4.1). Moreover, no process executes line 16. Hence, at each round $r$ after $t$, a process executes line 14 or line 17. We consider three cases.

- Case 1: All the processes that execute $r$, execute line 17.
  - So, all the processes set their estimates to a non-$⊥$ value. Due to Lemma 4.3, there are no more than $k$ different estimate values. Hence, all the processes that start $r + 1$ do it with at most $k$ different estimates, no one being equal to $⊥$. Due to Lemma 4.2 they decide.
- Case 2: During $r$ at least one process (but not all) executes line 17.
  - In that case, due to Lemma 4.3, each process $p_i$ that executes line 17 sets its est$_i$ local variable to a non-$⊥$ value taken from a set (namely, $PH2_{EST}(r)$) that includes at most $k$ non-$⊥$ values. The other processes execute the line 14. There is a probability (> 0) such that each of those processes sets its estimate variable to a non-$⊥$ value in $PH2_{EST}(r)$.
- Case 3: During $r$ no process executes line 17.
  - In that case, all the processes execute line 14. There is a probability (> 0) that they get no more than $k$ different estimate values (all different from $⊥$).

In the case 1, the termination is obtained. Let us consider the cases 2 and 3. During any round after $t$, there is a probability $p > 0$ that there are at most $k$ estimate values
each different from $\bot$. Hence, there is a probability $P(\alpha) = p + p(1-p) + p(1-p)^2 + \cdots + p(1-p)^{\alpha-1} = 1 - (1-p)^{\alpha}$ that, after at most $\alpha$ rounds, the processes have no more than $k$ estimate values, each different from $\bot$. As $\lim_{\alpha\to\infty} P(\alpha) = 1$, it follows that, with probability 1, all processes will start a round with no more than $k$ estimate values, each different from $\bot$. Then, according to Lemma 4.2, they will decide. □

4.3 Uniform Agreement

Theorem 4.5. No more than $k$ different values are decided.

Proof. Let $r$ be the first round during which some processes decide. They decide at line 16. Due to Lemma 4.3, the set $PH_2\text{EST}(r)$ contains at most $k$ non-$\bot$ values. Moreover, it follows from line 15, that a process that decides can only decide one of those values.

Let us now consider a process $(p_i)$ that proceeds to round $r + 1$. We claim (see its proof below) that its estimate $(est_i)$ is updated to a value of the set $PH_2\text{EST}(r)$ before it proceeds to $r + 1$. From this claim we conclude that a value $\notin PH_2\text{EST}(r)$ cannot be the value of an $est_j$ local variable after round $r$. Hence, any future decision at line 16 can only be a value of $PH_2\text{EST}(r)$. Moreover, a decision taken at line 4 does not increase the number of values decided at line 16.

Proof of the claim. Let $p_i$ be a process that decides at round $r$. From line 15, we conclude that $\bot \notin ph_2\text{REC}$. Moreover, the set $ph_2\text{REC}$ contains only values coming from the $ph_2\text{est}$ local variables of a majority of processes (line 12). Let $p_j$ be a process that progresses to $r + 1$. As $p_j$ has also received values from a majority of processes (line 12), we have $ph_2\text{REC}\cap ph_2\text{REC} \neq \emptyset$. Hence, $ph_2\text{REC} \neq \{\bot\}$. It follows that $p_j$ has not executed line 14 before progressing to $r + 1$. It has necessarily executed line 17. Consequently it has updated $est_j$ to a non-$\bot$ value of $PH_2\text{EST}(r)$. End of the proof of the claim. □

5. DISCUSSION

5.1 Cost of the Protocol

The cost of the protocol is the cost of the reliable broadcasts plus the cost of the task $T2$. In the task $T2$, the PHASE1 and PHASE2 messages carry only a round number and an estimate value. Each round requires two communication steps, and each communication step requires $2n$ broadcasts. In the most favorable case, the processes decide in a single round. This case occurs when the processes initially propose no more than $k$ different values.

5.2 An Improvement

When it executes line 14, a process $p_i$ can get the $\bot$ value, and consequently start a new round with $est_i = \bot$. This can prevent values different from $\bot$ to be the estimate of $W$ processes at the beginning of the next round, thereby delaying the decision.

A way to prevent this “bad” situation is to force any process $p_i$ to have an estimate value $est_i$ different from $\bot$ when it starts a new round. This can be obtained by replacing line 14 (namely, $est_i \leftarrow val_i[\text{random}]$) with the following sequence of statements:

\[
x \leftarrow \text{random};
\]

\[
\text{while } val_i[x] = \bot \text{ do } x \leftarrow (x \mod n) + 1 \text{ enddo;}
\]

\[
est_i \leftarrow val_i[x]
\]

5.3 On the $f < n/2$ Condition

The proposed protocol assumes $f < n/2$ (line 12). This condition is required to ensure the Uniform agreement property, as discussed in this section.

Let us consider a round $r$, and let $\Pi_1(r)$ (resp. $\Pi_2(r)$) be the set of processes participating in the first (resp. second) phase of $r$. As we have seen (Lemma 4.3, Section 4.1), the first phase builds a set $PH_2\text{EST}(r)$ such that $\cup_{p_i \in \Pi_1(r)} ph_2\text{REC} \subseteq PH_2\text{EST}(r) = \cup_{p_i \in \Pi_1(r)} \{ph_2\text{est}_i\}$.

This set includes at most $k$ different values, plus possibly the $\bot$ value.

Let us assume that $f \geq k$ and $f$ is not constrained by $f < n/2$. In that case, for not being blocked forever at line 12, a process has to wait for only $(n - f)$ PHASE2 messages. Let us consider the scenario where:

- $n = 2f$.
- There is a set $P$ of $(n - f) = f$ processes, such that any $p_i \in P$ starts the second phase of $r$ with $ph_2\text{est}_i \neq \bot$.
- There is a set $Q$ of $(n - f) = f$ processes, such that $Q \cap \emptyset = \emptyset$, and any $p_i \in Q$ starts the second phase of $r$ with $ph_2\text{est}_i = \bot$.
- Each process $p_i \in P$ receives PHASE2 messages from the $f$ processes of $P$. Consequently, for any of them $\bot \notin ph_2\text{REC}$. Hence, the processes of $P$ decide at lines 15-16. They can decide up to $k$ different values. Moreover, the $f$ processes of $P$ crash after deciding and their DEC messages take an arbitrary long time before being delivered to the processes of $Q$.
- Each process $p_i \in Q$ receives PHASE2 messages from the $f$ processes of $Q$. Consequently, they have $ph_2\text{REC} = \{\bot\}$. Hence, each $p_i$ executes line 14 and selects randomly a new estimate for the next round. It is possible that those $f$ estimates are different from the $k$ values decided by the processes of $P$. Hence, during the next rounds, the $f$ processes of $Q$ can decide values different from the ones decided by the processes of $P$.

The previous scenario is due to the fact that the processes of $Q$ are not aware of which are the values decided by the processes of $P$. In terms of consensus protocols [4], the values decided by the processes of $P$ have to be “locked”. This means that, at any round, the lines 14 and 15-16 have to be mutually exclusive: if some processes execute lines 15-16, no process can execute line 14 and vice versa. This exclusion can be realized by ensuring that at any round the following condition is satisfied (quorum-based approach developed in [13]):

\[
\forall (p_i, p_j) : \text{ph}_2\text{REC} \cap \text{ph}_2\text{REC} \neq \emptyset.
\]

This condition guarantees that when a round a process executes lines 15-16, the others execute either the same lines or line 17. The previous condition on the $\text{ph}_2\text{REC}$ sets is satisfied by demanding that each process waits for PHASE2 messages from a majority of processes. Hence, the constraint $f < n/2$ is required to ensure that no more than $k$ values are
decided. Said in another way, the locking of decided values requires a “non-partitioning” guarantee to cope with the case where processes do not decide during the same round.

5.4 On the \( R \leq (n - f) \) Condition

The protocol requires the condition \( R \leq (n - f) \) to ensure that a correct process will enter a round during which it will receive the same value from \( W = [(n + 1)/(k + 1)] \) processes. Hence, differently from the \( f < n/2 \) condition, this condition is related to the Termination property.

As we have seen, the condition \( R \leq (n - f) \) translates as \( f < n - k|n/(k + 1)| \). For \( k = 1 \) (consensus) it becomes \( f < n/2 \) which has been shown to be a necessary requirement to solve the consensus problem with randomization [2]. Let \( P(\alpha) \) be the protocol proposed in Figure 1 where \( k \) has been replaced with \( \alpha \). Let us notice that, as any solution to the consensus problem solves (in a brute way) the \( k \)-set agreement problem, it is possible to use \( P(1) \) to solve the \( k \)-set agreement problem, whatever \( k \). Moreover, from \( f < n - k|n/(k + 1)| \), we conclude that \( f \) decreases when \( k \) increases. This means that, whenever \( k > 1 \), \( P(1) \) tolerates more faults than \( P(k) \). At first sight, this appears rather counter-intuitive and disturbing, as an increase of \( k \) (the maximum number of values that can be decided) “should” allow for more faults. Actually, this is not counter-intuitive for the following reason related to the randomized part of the protocol and consequently to its expected convergence speed.

As the \( k \)-set agreement problem can be solved without the help of additional assumptions when \( f < k \), the proposed protocol is useful only when \( f \geq k \) (C1). When combining with C1 the constraint under which the protocol works, namely \( f \leq n - k|n/(k + 1)| - 1 \) (C2), we get the condition \( n > k(k + 1) \). This means that if \( n \leq k(k + 1) \) then there is no triple \((n, k, f)\) that satisfies the required constraints C1 and C2. For a given value of \( n \), let \( K \) denote the greatest value of \( k \) that satisfies the condition \( n > k(k + 1) \). The two following curves describe a tradeoff between the maximum number of faults and the convergence speed of the protocol.

- The curve depicted in Figure 2.a represents the maximum value of \( f \) according to the value of \( \alpha \), \( 1 \leq \alpha \leq K \). As already noted, \( P(1) \) tolerates more faults than \( P(2) \), etc. According to \( n, f \) and \( \alpha \), this curve defines the domain where the protocol can work and is useful (grey area).

- It has been shown (Lemma 4.2) that for \( P(\alpha) \) to terminate during a round \( r \), whatever the actual communication pattern during \( r \), the processes must have no more than \( \alpha \) different estimate values when they start \( r \). Let us consider the case where the \( (y) \) processes that execute a round \( r \), do execute line 14. The curve depicted in Figure 2.b represents the probability \( p_y(\alpha) \) to have no more than \( \alpha \) different estimate values at the end of \( r \); \( p_y(\alpha) \) is equal to \((\alpha/n)\)\(^y\).

Let us now consider a particular value \( \alpha = k \) (see the figure). Each protocol \( P(\alpha) \) \( 1 \leq \alpha \leq k \), can solve the problem provided that \( f < n - \alpha|n/(\alpha + 1)| \). Let \( \alpha_1 \) and \( \alpha_2 \) be such that \( 1 \leq \alpha_1 < \alpha_2 \leq k \). Figure 2.a shows that \( P(\alpha_1) \) allows more faults than \( P(\alpha_2) \), whereas Figure 2.b shows that \( P(\alpha_2) \) can a priori converge more quickly than \( P(\alpha_1) \). Hence, the protocol presented in Section 3 provides an interesting tradeoff relating the convergence speed with the maximum number of tolerated faults. More choices allow either more faults or more efficient runs.

6. CONCLUSION

In the \( k \)-set agreement problem, the processes propose values and each correct process has to decide a value such that a decided value is a proposed value and there are no more than \( k \) different decided values. This problem has no solution when \( f \) (the maximum number of processes that can crash) is \( \geq k \).

To circumvent this impossibility result, we have investigated in a previous work the concept of limited accuracy failure detectors [14]. Here, we have investigated the randomization approach. A randomized \( k \)-set agreement proto-
col has been proposed. This protocol does not require the a priori knowledge of the set of values initially proposed by the processes. It is based on a simple combination of randomization (to ensure the convergence) and reliable broadcast (for the processes to disseminate the values they propose). The protocol requires that at least a majority of processes be correct and exhibits an interesting tradeoff relating its expected convergence speed with the number of faults it tolerates. To our knowledge, this protocol is the first randomized $k$-set agreement protocol proposed so far. Moreover, combining the proposed protocol with [14] would provide a hybrid $k$-set agreement protocol.

The following two problems remain open: (1) “Does it exist a randomized protocol that solves the $k$-set agreement problem despite $f \geq k$, without requiring the ‘majority of correct processes’ assumption?” (2) “Does it exist a protocol for which more choices allow both more faults and more efficient runs?”

7. REFERENCES