HAL
open science

Unreliable Distributed Timing Scrutinizer to Converge toward Decision Conditions

Emmanuelle Anceaume, Eric Mourgaya, Philippe Raïpin-Parvédy

To cite this version:

Emmanuelle Anceaume, Eric Mourgaya, Philippe Raïpin-Parvédy. Unreliable Distributed Timing Scrutinizer to Converge toward Decision Conditions. Studia Informatica Universalis, 2008, 6 (2), pp.23–50. hal-00916690

HAL Id: hal-00916690

https://hal.science/hal-00916690v1

Submitted on 16 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Unreliable Distributed Timing Scrutinizer to Converge toward Decision Conditions

Observateurs de temps distribués, conditions de décision

E. Anceaume — E. Mourgaya — P. R. Parvédy

ABSTRACT. *In this paper, we propose to extend the condition-based approach introduced and developed by Mostéfaoui et al. in 2001 by characterizing $\mathscr{V}_f^n$, the set of all the possible input vectors containing the values proposed by n processes. The condition-based approach consists in identifying sets of input vectors for which the consensus is directly solvable (i.e., in one communication step) in a pure asynchronous model despite up to f crashes. We focus on all the other input vectors. Among them, we identify those that allow to solve the consensus problem in two communication steps still in a pure asynchronous model. For the other ones, we rely on a distributed oracle that enables the input vectors to converge toward a good patterned vector with probability one. We specify a protocol that benefits from this approach to solve the consensus problem very simply and efficiently.*

RÉSUMÉ. *Nous proposons d'étendre l'approche conditionnée, introduite et développée par Mostéfaoui et al. en 2001, par caractérisation de l'ensemble $\mathscr{V}_f^n$ de tous les vecteurs entrants possibles contenant les valeurs générées par n processus. L'approche conditionnée consiste à identifier les ensemble de vecteurs entrants pour lesquels le consensus est directement résoluble (c-a-d en une seule étape) dans un modèle asynchrone pur en dépit de f crashes. Nous nous intéressons aux autres vecteurs entrants parmi lesquels nous identifions ceux qui permettent de résoudre le consensus en deux étapes de communications dans un modèle asynchrone pur. Pour les autres vecteurs, tout dépend d'un oracle distribué qui fait converger les vecteurs vers un vecteur bien conditionné avec probabilité 1 . Nous spécifions un protocole qui tire avantage de cette approche pour résoudre simplement et efficacement le consensus*

KEYWORDS: *Asynchronous systems, condition-based approach, converging-based approach, consensus problem.*

MOTS-CLÉS : *Système asynchrone, approche conditionnée, Problème d'agrément, consensus distribué, crashes, oracle*

1. Introduction

Agreement problems are unavoidable in distributed systems. Such problems arise whenever the processes of the system have to reach a common decision, or to share a view of the computation progress. All the agreement problems can be reduced to the same basic problem, namely the consensus problem (Shostak *et al.,* 1981). In this problem, processes propose a value and must unanimously and irrevocably decide on the same value that is related to the proposed values. Unfortunately, Fisher, Lynch, and Paterson (Fisher *et al.,* 1985) demonstrated that whenever the distributed system is both *asynchronous* and prone to *crash failures* there is no deterministic solution to the consensus problem (in the following, this result is referred as the FLP result). Several approaches have been proposed to circumvent this impossibility result (Randomized protocols have been proposed (e.g., (Ben-Or, 1983)), weaker specifications of the problem have been given leading for example to the k-set agreement (Chaudhuri, 1993), or to the approximate agreement (Dolev *et al.,* 1986), or properties have been given in the failure detection quality (Chandra *et al.,* 1996)). Recently, another approach based on conditions has been investigated (Mostéfaoui *et al.,* 2001a, Mostéfaoui *et al.,* 2001b, Mostéfaoui *et al.,* 2001c, Mostéfaoui *et al.,* 2002). Contrary to the former one, this approach does not try to circumvent the FLP result (Fisher *et al.,* 1985). Rather it finds restrictions on the set of input vectors (the i-th entry of an input vector contains the value proposed by process p_i, with $\mathscr{V}$ the set of possible values) for which the problem becomes solvable. In MostÈfaoui et al (Mostéfaoui *et al.,* 2001a, Mostéfaoui *et al.,* 2001b, Mostéfaoui *et al.,* 2001c, Mostéfaoui *et al.,* 2002), the authors focus on conditions C that identify sets of vectors of $\mathscr{V}^n$ that allow n processes to directly solve the consensus problem in spite of up to f crashes in the standard asynchronous model. Such conditions are called f-acceptable conditions, and denoted $\mathscr{C}_f^{[e]}$ with e the distance between these vectors (Mostéfaoui *et al.,* 2001a). These conditions have to meet constraints that are defined in terms of a predicate P and a function S. Both P and S have to satisfy some termination, agreement and validity properties that guarantee that for any two input vectors satisfying an f-acceptable condition, then the consensus problem is solved. As shown in (Mostéfaoui *et al.,* 2003),

this approach is very efficient when the probability of process crashes is low (a common fact in practice).

Clearly among all the input vectors belonging to $\mathscr{V}^n$, only a subset of them belongs to f-acceptable conditions. Otherwise, it would contradict the FLP result (Fisher *et al.,* 1985). For the other ones, consensus protocols need to rely on any combination merging random oracle, leader oracle or unreliable failure detector to guarantee the termination property (Mostéfaoui *et al.,* 2002).

In this paper, we focus on those input vectors that do not satisfy f-acceptable conditions. Clearly, among them, some are very close to good patterned vectors while the others are not. Specifically, information contained in the former ones is sufficiently significant to guarantee that with a high probability all the correct processes highlight the same value from the received input vector. Thus, by proposing this value as new input value, all the processes converge very quickly toward the same decision value. By relying on such a property, one can build very simple and efficient consensus protocols that decide in two rounds of computation despite up to f crash failures, and without any help (i.e., without any oracles). This paper proposes a characterization of this set of vectors. In the following, this set of vectors is denoted $\mathscr{C}_f^{[e],s}$. For all the other vectors, denoted hereafter $\mathscr{C}_f^{[e],w}$, that is, those which are far from the good patterned ones, it is necessary to enrich or refine their information. Possible solutions are to wait for more input values (i.e., more than $n - f$ input values) or to propose a new input value that has some chance to be proposed by other processes. Clearly the first solution is impossible in a pure asynchronous model essentially because one cannot make the difference between a very slow process and a crashed one. Concerning the second solution, the probability that all correct processes propose the same new input value is very low simply because their input vectors may be different. Our idea is to combine both solutions. To this end, we rely on a timing oracle, the unreliable distributed timing scrutinizer (Anceaume *et al.,* 2002), that provides infinitely often the actual time needed to receive x messages provided that x processes have not crashed, with $x \geq n - f$. By relying on this mechanism, each process may enrich the information contained in its input vector, and thus, can refine the value returned by the decision

function S (Mostéfaoui *et al.,* 2001a). By proposing this value as new input value, the probability to converge toward a good patterned vector (i.e., a vector that satisfies an acceptable condition) increases to eventually be equal to 1. Clearly, one could have relied on any oracle (random oracle, leader oracle or unreliable failure detector) as in MostÈfaoui et al. (Mostéfaoui *et al.,* 2002). However, their approach guarantees an eventual termination or a termination with probability 1, while the $UDTS$ mechanism ensures infinitely often a termination in a bounded number of steps. We show in this paper that there exists for a given $e \in [0, f]$ a strict partition of the set of all the possible vectors $\mathscr{V}_f^n$ in $\mathscr{C}_f^{[e]}$, $\mathscr{C}_f^{[e],s}$ and $\mathscr{C}_f^{[e],w}$.

We present experimental results that emphasize the interest of the proposed approach. For a given f-acceptable condition $\mathscr{C}_f^{[e]}$ and a given $\mathscr{V}$, we present the distribution of vectors in $\mathscr{C}_f^{[e]}$, in $\mathscr{C}_f^{[e],s}$, and in $\mathscr{C}_f^{[e],w}$. These results show that for a binary consensus, the ratio of vectors that satisfy $\mathscr{C}_f^{[f]}$ decreases with increasing values of n (e.g., for $n = 3$, 40% of vectors satisfy $\mathscr{C}_f^{[f]}$, while for $n = 9$, 5% of them satisfy $\mathscr{C}_f^{[f]}$). This result is reversed for the vectors that satisfy $\mathscr{C}_f^{[f],s}$ (for $n = 3$, 45% of vectors satisfy $\mathscr{C}_f^{[e],s}$, while for $n = 9$, 53% of them satisfy $\mathscr{C}_f^{[f],s}$).

The contributions of this paper are as follows:

– We propose, for a given acceptable condition $\mathscr{C}_f^{[e]}$ and a probability $k \in (0.5, 1]$, a characterization of the set $\mathscr{V}_f^n$. This characterization includes the set of vectors that satisfy $\mathscr{C}_f^{[e]}$ defined by MostÈfaoui et al. (Mostéfaoui *et al.,* 2001a, Mostéfaoui *et al.,* 2001b, Mostéfaoui *et al.,* 2001c, Mostéfaoui *et al.,* 2002), and the two above sets of vectors $\mathscr{C}_f^{[e],s}$ and $\mathscr{C}_f^{[e],w}$. The set $\mathscr{C}_f^{[e],s}$ contains all the input vectors for which the probability to converge toward a good patterned vector is greater than or equal to k. The set $\mathscr{C}_f^{[e],w}$ contains all the vectors whose information is not significant enough to deterministically highlight some value.

– We illustrate our approach by providing, for a given f-acceptable condition, the distribution of vectors in each of the classes $\mathscr{C}_f^{[f]}$, $\mathscr{C}_f^{[f],s}$

and $\mathscr{C}_f^{[f],w}$, according to n and $|\mathscr{V}|$.

– We specify a deterministic consensus protocol that benefits from the condition-based approach together with the characterization of $\mathscr{V}_f^n$ to augment the number of input vectors that lead to a decision value either directly (condition-based approach) or very quickly (in two rounds of protocol). For all the other vectors, the protocol relies on the UDTS oracle to decide in a bounded number of steps.

The remaining of the paper is organized as follows. Section 2 describes the computational model and the consensus problem. Section 3 outlines the condition-based approach. Section 4 presents the convergence-based approach, the approach we propose. Section 5 presents the deterministic consensus protocol. Section 6 concludes.

2. The Distributed Computation Model and the Consensus Problem

2.1. *The Distributed Computation Model*

The system consists of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, \ldots, p_n\}$. We denote by $\langle ij \rangle$, the link from p_i to p_j. Processes communicate and synchronize by sending and receiving messages. The network can be represented as a complete graph structure $G = \{\Pi, E\}$, where $E = \{\langle ji \rangle,\ i, j \in [1, n]^2\}$. For each process p_i we denote by $G_i = \{\Pi, E_i\}$ the sub-graph defined by a central node p_i and the set $E_i = \{\langle ji \rangle, j \in [1, n]\}$. In the following we call this sub-graph structure a *star-graph* (example in Figure 1).

2.1.1. **Failure Model**

The execution of a process might stop prematurely (crash failure), or can fail to process some code in a timely manner (performance failure). A crashed process does not recover, contrary to a process that suffers from performance failures. By definition, a *correct* process is a process that never crashes, while a *faulty* one is not correct. The maximum number of faulty processes is denoted f. Both correct and faulty processes can suffer from performance failures.

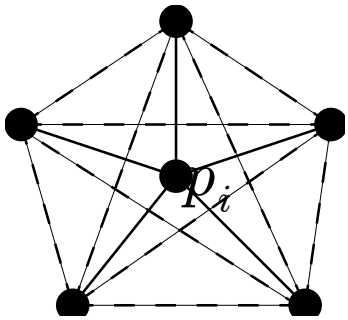

Figure 1: *star-graph* of p_i

The datagram service is assumed to be completely controlled by an adversary which has all freedom to delay (but not to drop or to corrupt) messages. In other words, the datagram service has a performance failure semantic (*i.e.*, it can fail to deliver messages in a timely manner).

Finally, we assume that each non-crashed process has access to a correct hardware clock H_p, *i.e.*, with a known and finite drift rate. This assumption simplifies applications since they have to deal with crash failures anyhow but they do not have to deal with faulty clocks like "fast" or "slow" clocks (Cristian *et al.,* 1999). Note that this hypothesis is not unrealistic even in large networks. For example, GPS (Global Positioning System) clocks are becoming affordable, and they can provide clocks whose drift is very small and known (Verissimo *et al.,* 2000). In case the knowledge of local drifts cannot be ensured, it is possible to use past heartbeat messages to get accurate estimates of the expected arrival times of future heartbeats, and then to use them to synchronize processes. Using past heartbeat messages does not involve any additional overhead in messages.

2.1.2. **Behavior of the Communication Links**

The behavior of communication links follows the one described in (Anceaume *et al.,* 2002) (example in Figure 2). Briefly, transmission

delays are modeled by a random variable $\mathscr{X}$. The behavior of communication links is encompassed by the *covering* and *locality* properties (Anceaume *et al.,* 2002). The *covering* property guarantees that the behavior of message transmission delays over the communication link can always be approximated by a sequence of probabilistic laws. While the *locality* property guarantees that any two consecutive elements of this sequence are similar. More precisely, since $\mathscr{X}$ does not follow a known classical probabilistic law, we cover the history of message transmission delays by a sliding window on which $\mathscr{X}$ follows a known classical probabilistic law.

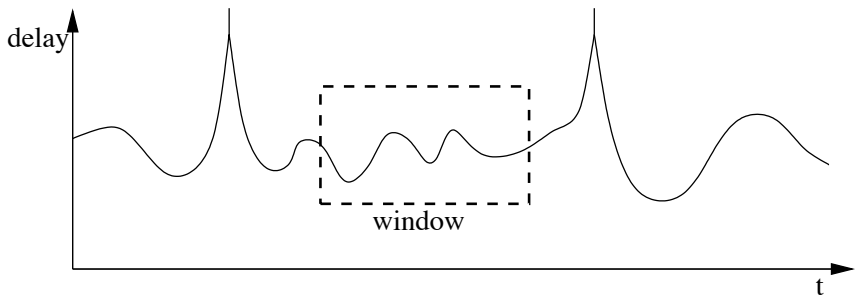

Figure 2: example of transmission delays of a particular link according to time

A communication link is *stable* if it satisfies the *locality* and *covering* properties. In the following, we assume that the network is *stable*, i.e., any link connecting any two correct processes is stable.

2.1.3. **The Unreliable Distributed Timing Scrutinizer**

The unreliable distributed timing scrutinizer (UDTS) is defined in (Anceaume *et al.,* 2002). This mechanism captures the state of the network and, based on its observation, estimates messages transmission delays. This is a distributed module such that each process has access to a local UDTS. The local UDTS associated to process p_i is denoted $UDTS_i$. Upon p_i request, $UDTS_i$ predicts an estimate Δ_i of the waiting time needed to receive a given number of messages over G_i. Due to the possible unpredictable but transient fluctuations over G_i, Δ_i can be temporarily under-estimated or over-estimated. However, by peri-

odically updating its estimate, $UDTS_i$ quickly and easily adapts to G_i behavior. It is implementable and by capturing the network properties, one may determine the waiting time that minimizes the number and/or the duration of computational rounds of the agreement protocol (Anceaume *et al.,* 2002), or may simply compute the waiting needed time to receive $m, m+1, .., m'$ messages, with $1 \leq m, m' \leq n$. This is this latter feature that we use in this paper. More precisely, when p_i queries $UDTS_i$ with an interval of values $[\lceil k_0 n \rceil, n]$ with $\lceil k_0 n \rceil$ the smallest number of messages needed to guarantee the safety property of the agreement protocol run by p_i[1], $UDTS_i$ returns a vector $TimEst_i$ of estimated waiting times. This vector contains $n - \lceil k_0 n \rceil + 1$ entries. $TimEst_i[k]$ is an estimation of the waiting time needed to receive k messages over G_i if $UDTS_i$ is able to compute this time, that is if at least the k corresponding links are stable, otherwise $TimEst_i[k]$ is equal to $\perp$. An Unreliable Distributed Timing Scrutinizer ($UDTS$) is completely defined by the *Completeness* property. This property characterizes the detection of stable links, and is defined as follows. For all $i \in [1, n]$:

– *Completeness property*: all stable links of G_i are detected.

As previously said, the $UDTS$ oracle is unreliable in the sense that its estimations can be temporarily under-estimated or over-estimated. However, by the stable network and the completeness properties, the returned estimation is infinitely often correct (Anceaume *et al.,* 2002). So the greatest non $\perp$-value of vector $TimEst_i$ allows infinitely often the receipt of all the messages sent to p_i. The proposed consensus algorithm relies on such a property to make any input vector in $\mathscr{C}_f^{e,w}$ converge to $\mathscr{C}_f^{e,s}$. This is detailed hereafter.

2.2. *The Consensus Problem*

The consensus problem can be informally defined as follows : Each process that executes the consensus proposes a value, and all the correct processes must eventually decide on the same initial value. Formally the consensus problem satisfies the following three properties :

1. Typically, $\lceil k_0 n \rceil$ is equal to $\lceil \frac{n+1}{2} \rceil$ or to $\lceil \frac{2n+1}{3} \rceil$ in agreement protocols

– *Termination:* Every correct process eventually decides exactly one value.

– *Validity:* If a process decides v, then v was proposed by some process.

– *Agreement:* If a process decides v, then all correct processes eventually decide v.

We introduce the Finite Termination property. This property guarantees that correct processes terminate infinitely often in a bounded number of computation steps.

3. The Condition-Based Approach

As said in the Introduction, this work relies on the condition-based approach which has been introduced and developed by MostÈfaoui and al.(Mostéfaoui *et al.,* 2001b), (Mostéfaoui *et al.,* 2001c), (Mostéfaoui *et al.,* 2001a). This section outlines this approach.

3.1. *The Approach*

The condition-based approach consists in considering conditions that make the consensus problem solvable, despite up to f crashes. More precisely, it consists in identifying sets of input vectors for which it is possible to design a protocol that does not require additional assumptions, i.e., a pure asynchronous environment. An input vector I is a size n vector, whose i-th entry contains the value proposed by process p_i, or $\perp$ if p_i did not take any step in the execution. The set of values that can be proposed by the processes is denoted $\mathscr{V}$. Note that $\perp \notin \mathscr{V}$. If at most f processes can crash, at most f entries are equal to $\perp$. These input vectors are called views. For $I \in \mathscr{V}^n$, with $\mathscr{V}^n$ the set of all possible input vectors with all entries in $\mathscr{V}$, then $\mathscr{I}_f$ denotes the set of possible views J such that I agrees with J in all the non $\perp$-entries of J. By extension, $\mathscr{V}_f^n$ denotes the set of all possible vectors with at most f entries equal to $\perp$. To solve consensus, a condition has to meet constraints that are defined in terms of a predicate P and a function S. Both P and S have to satisfy a termination, validity, and agreement properties. Informally, these properties say that predicate P allows a process p_i to test

whether a decision value can be computed from its view, while function S applied on a valid view allows p_i to get as decision value an initial value.

– $T_{C\rightarrow P}$ (Termination property): $I \in C \Rightarrow \forall J \in \mathscr{I}_f : P(J)$

– $V_{P\rightarrow S}$ (Validity property): $\forall J \in \mathscr{I}_f : P(J) \Rightarrow S(J) =$ a non-$\perp$ value of J

– $A_{P\rightarrow S}$ (Agreement property): $\forall J1, J2 \in \mathscr{I}_f : (J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

The latter one introduces the ordering property $(J1 \leq J2)$ for $J1, J2 \in \mathscr{V}_f^n$ which is defined as follows: $\forall k : J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$. This ordering property strongly relies on the invocation to the snapshot primitive. This primitive is assumed to be available to all the processes in a standard asynchronous shared-memory system. However in a message-passing model, as assumed in this paper, there is no snapshot-like primitive. However, in such a model, one can easily implement the comparison ordering property $J1 \sim^e J2$. This ordering property can be defined as follows: Let $J1, J2 \in \mathscr{V}_f^n$ be any two input vectors. Let $e \in [0, 2f]$ with $f < \frac{n}{2}$.

Definition 1 *$J1$ is $e-$comparable to $J2$, denoted $J1 \sim^e J2$, iff for all $i \in [1..n]$ we have*
$(J1[i] = J2[i] \vee J1[i] = \perp \vee J2[i] = \perp)$ and at most e t-uplets $(J1[i], J2[i])$ with $J1[i] = \perp \vee J2[i] = \perp$.

Clearly the comparison property $J1 \sim^e J2$ is weakest than the ordering property $(J1 \leq J2) \vee (J2 \leq J1)$. Hence we need to weaken, as MostÈfaoui and al. (Mostéfaoui *et al.,* 2001c), the agreement property of an acceptable condition as follows :

– $A^{[e]}_{P\rightarrow S}$ (e-Agreement property): $\forall I \in \mathscr{V}^n : P(J1) \wedge P(J2) \wedge (J1 \sim^e J2) \Rightarrow S(J1) = S(J2)$

A condition is f-acceptable if there exists a predicate $P^{[e]}$ and a function S satisfying the Termination, Validity and e-Agreement properties. MostÈfaoui et al. (Mostéfaoui *et al.,* 2001a) characterize the largest set of conditions that allows to solve the consensus problem. Henceforth,

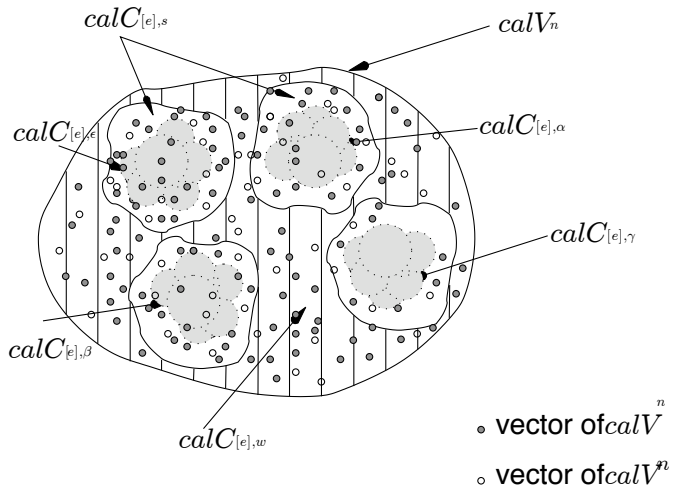

Figure 3: Characterization of $\mathscr{V}_f^n$ for a Given Condition $C_f^{[e]}$ and for a given k

we consider only f-acceptable conditions. We denote these conditions by $\mathscr{C}_f^{[e]}$ where e can be seen as the degree of asynchrony of the system.

3.2. *The Condition-based Module*

The *condition-based* module is a distributed module which implements the condition-based approach, more precisely the acceptability parameters $P^{[e]}$ and S of the given acceptable condition $\mathscr{C}_f^{[e]}$ (Mostéfaoui *et al.,* 2001a). Processes exchange their estimate values to define their current views V_i of proposed values, and then apply predicate P to V_i, and accordingly function S returns the decision value if predicate P returns true, otherwise S returns a $\perp$-value, that is a non-decision value. To guarantee the termination of a condition-based consensus protocol, one needs to associate the condition-based module with any combination merging random oracle, leader oracle or unreliable failure detector oracles (Mostéfaoui *et al.,* 2001a).

4. The Convergence-based Approach

One can depict $\mathscr{V}_f^n$ as a collection of decision sets isolated from each other by a non-decision area. Each isolated decision set contains all the input vectors J that satisfy predicate $P^{[e]}$ and such that $S(J) = a$. The non-decision area $\mathscr{V}_f^n/\mathscr{C}_f^{[e]}$ contains all the input vectors J' such that $\neg P^{[e]}(J')$. This is illustrated in Figure 3. As briefly presented in the Introduction, our approach, that we call the convergence-based approach, consists in characterizing this non-decision area. Clearly due to FLP result (Fisher *et al.,* 1985), $\mathscr{V}_f^n/\mathscr{C}_f^{[e]}$ is a non-empty set. Thus the convergence-based approach consists in characterizing the set $\mathscr{V}_f^n/\mathscr{C}_f^{[e]}$ with $f \geq 1$. For a given f-acceptable condition, and a value $k \in (0.5, 1]$, we propose to split the non-decision area $\mathscr{V}_f^n/\mathscr{C}_f^{[e]}$ into two sub-areas $\mathscr{C}_f^{[e],s}$ and $\mathscr{C}_f^{[e],w}$. Informally, the former one, that we call the strong agreement area, contains all the input vectors such that with a high probability converge toward a perfectly determined value, while the second one contains vectors such that with a low probability converge toward a perfectly determined value. Note that in the following we use indifferently the term convergence area and convergence set.

4.1. *Notations and Definitions*

Given $e \in [0, 2f]$, $k \in (\frac{1}{2}, 1]$, and an input vector $J \in \mathscr{V}_f^n$, predicate $P_k^{[e]}(J)$ allows a process p_i to test whether for any e-comparable input vector $J1 \in \mathscr{V}_f^n$, the probability that $S(J) = S(J1)$ is greater than or equal to k :

– $P_k^{[e]}(J) : \forall J1 \in \mathscr{V}_f^n / J \sim^e J1, Pr[S(J) = S(J1)] \geq k$

where $Pr[X]$ denotes the probability that event X occurs.

Within this context, we can define the notions of strong agreement set $\mathscr{C}_f^{[e],s}$ and weak agreement set $\mathscr{C}_f^{[e],w}$ as follows :

Definition 2 *Given an f-acceptable condition, we have:*

– $\mathscr{C}_f^{[e],s}$ *(Strong convergence set)* = $\{J \in \mathscr{V}_f^n / P_k^{[e]}(J) \wedge \neg P^{[e]}(J)\}$

– $\mathscr{C}_f^{[e],w}$ *(Weak convergence set)* = $\{J \in \mathscr{V}_f^n / \neg P_k^{[e]}(J) \wedge \neg P^{[e]}(J)\}$

Lemma 4.1 $\mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],s} = \mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],w} = \mathscr{C}_f^{[e],s} \cap \mathscr{C}_f^{[e],w} = \emptyset$

Proof We only prove the first part of the lemma, i.e., $\mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],s} = \emptyset$, since the proof of the other two parts are very similar.

The proof is done by contradiction. Suppose that there exists some vector $I \in \mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],s}$. Then, by definition of $\mathscr{C}_f^{[e],s}$ and $\mathscr{C}_f^{[e],w}$, vector I satisfies $P^{[e]}(I)$ and $\neg P^{[e]}(I) \wedge P_k^{[e]}(I)$. Clearly, this is impossible. Thus vector I does not exist, and $\mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],s} = \emptyset$. $\square_{Lemma\ 4.1}$

The following lemma shows that $\{\mathscr{C}_f^{[e]}, \mathscr{C}_f^{[e],s}, \mathscr{C}_f^{[e],w}\}$ is a partition of $\mathscr{V}_f^n$, for a given $e \in [0, f]$ and $k \in (0.5, 1]$.

Lemma 4.2 $\mathscr{V}_f^n = \mathscr{C}_f^{[e]} \cup \mathscr{C}_f^{[e],s} \cup \mathscr{C}_f^{[e],w}$ *and* $\mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],s} \cap \mathscr{C}_f^{[e],w} = \emptyset$

Proof By lemma 4.1, the assertion $\mathscr{C}_f^{[e]} \cap \mathscr{C}_f^{[e],s} \cap \mathscr{C}_f^{[e],w} = \emptyset$ is trivially true.

We now prove that $\mathscr{V}_f^n = \mathscr{C}_f^{[e]} \cup \mathscr{C}_f^{[e],s} \cup \mathscr{C}_f^{[e],w}$. By definition, we have $\mathscr{C}_f^{[e]} \subset \mathscr{V}_f^n$, $\mathscr{C}_f^{[e],s} \subset \mathscr{V}_f^n$ and $\mathscr{C}_f^{[e],w} \subset \mathscr{V}_f^n$. Thus $\mathscr{C}_f^{[e]} \cup \mathscr{C}_f^{[e],s} \cup \mathscr{C}_f^{[e],w} \subseteq \mathscr{V}_f^n$. Let $I \in \mathscr{V}_f^n$. If $P^{[e]}(I)$ then $I \in \mathscr{C}_f^{[e]}$, otherwise we have $\neg P^{[e]}(I)$. If $P_k^{[e]}(I)$ then $I \in \mathscr{C}_f^{[e],s}$ otherwise $I \in \mathscr{C}_f^{[e],w}$. Thus $I \in \mathscr{C}_f^{[e]} \cup \mathscr{C}_f^{[e],s} \cup \mathscr{C}_f^{[e],w}$. $\square_{Lemma\ 4.2}$

4.2. *Properties*

Given an acceptable condition $\mathscr{C}_f^{[e]}$ with $P^{[e]}$ and S satisfying the properties $T_{C\to P}$, $V_{P\to S}$ and $A_{P\to S}^{[e]}$, we define the property $A_{P_k\to S}^{[e]}$. This property guarantees the agreement among all vectors in $\mathscr{C}_f^{[e],s}$:

– $A_{P_k\to S}^{[e]}$ (Strong Convergence Agreement property) :
$\forall I \in \mathscr{V}^n : \forall J_1, J_2 \in \mathscr{I}_f, P_k^{[e]}(J1) \wedge P_k^{[e]}(J2) \wedge (J1 \sim^e J2) \Rightarrow S(J1) = S(J2)$

Definition 3 *A strong agreement area is acceptable iff property* $A^{[e]}_{P_k \rightarrow S}$ *is guaranteed.*

A first consequence of this property is that $\mathscr{C}_f^{[e],s}$ can be partitioned in $|\mathscr{V}|$ subsets $\mathscr{C}_f^{[e],s,a}$ such that $\mathscr{C}_f^{[e],s,a} = \{J \in \mathscr{C}_f^{[e],s} / S(J) = a\}$ with $a \in \mathscr{V}$. We have $\mathscr{C}_f^{[e],s} = \bigcup_{a \in \mathscr{V}} \mathscr{C}_f^{[e],s,a}$ with $\bigcup_{a \neq b \in \mathscr{V}} (\mathscr{C}_f^{[e],s,a} \cap \mathscr{C}_f^{[e],s,b}) = \emptyset$. The second consequence is that if for some vector $J1$ validating predicate $P^{[e]}$ and such that the decision value is $S(J1)$ then for any other comparable vector $J2$ validating predicate $P_k^{[e]}$, then $J2$ converges also to $S(J1)$:

– $A^{[e]}_{P_k, P \rightarrow S}$ (Convergence Agreement property) :
$\forall I \in \mathscr{V}^n : \forall J_1, J_2 \in \mathscr{I}_f, P^{[e]}(J1) \wedge P_k^{[e]}(J2) \wedge (J1 \sim^e J2) \Rightarrow S(J1) = S(J2)$

4.3. *The Condition-Based*$^+$ *Module*

The *condition-based*$^+$ module is an extension of the *condition-based* module (see Section 3.2). It implements for a given f-acceptable condition $\mathscr{C}_f^{[e]}$ the predicate $P^{[e]}$ and function S as in MostÈfaoui et al. (Mostéfaoui *et al.,* 2001c) and predicate $P_k^{[e]}$. Figure 4 describes the algorithm of the condition-based$^+$ module. In addition to the implementation of the predicates $P^{[e]}$ and $P_k^{[e]}$ and function S, this module computes the distance which separates vector V_i from the nearest vector $J \in \mathscr{C}_f^{[e]}$ if $V_i \in C_f^{[e],s}$ and from the nearest $J \in \mathscr{C}_f^{[e],s}$ if $V_i \in C_f^{[e],w}$. The distance which separates vector V_i from the nearest vector J in $\mathscr{C}_f^{[e]}$ or in $\mathscr{C}_f^{[e],s}$ is equal to the minimum number of estimates needed to possibly decide or strongly converge. This distance is used by p_i to estimate whether it is worth waiting for more estimates or not. The distance from V_i to a set F is computed as follows:
d(V_i, F) = min($\{$distance(V_i, J), $J \in F\}$) with distance(V_i, J) the number of couples $(V_i[j], J[j])$ that contain exactly one $\perp$-value.

Function $\texttt{GetCond}^{+}(V)$

(1) **if** $\neg(P_k^{[e]}(V) \vee P^{[e]}(V))$ **then**
(2) **return** $\texttt{PutCond}^{+}(S(V), d(V, C_f^{[e],s}), false, false)$;
(3) **else**
(4) **return** $\texttt{PutCond}^{+}(S(V), d(V, C_f^{[e]}), P^{[e]}(V), P_k^{[e]}(V))$;

Figure 4: The *Condition-based*$^+$ Module Run by Process p_i

4.4. *Some Experimental Results*

This section provides a quantitative evaluation of the proportion of vectors within the decision, the strong convergence and the weak convergence areas. This evaluation is done by considering the f-acceptable condition $\mathscr{C}2_f^{[f]}$ proposed by MostÈfaoui et al (Mostéfaoui *et al.,* 2001c). Since we assume a message passing model we need to consider f-comparable input vectors, with $f < \frac{n}{2}$. Predicate $P2_f^{[f]}$ and function $S2_f$ are respectively defined as follows. Let J be an input vector belonging to $\mathscr{V}_n^f$. Then, $P2_f^{[f]}(J) \equiv \#_{1st}(J) - \#_{2nd}(J) > f + f - \#_{\perp}(J)$ (with $\#_{1st}(J)$ and $\#_{2nd}(J)$ denote the occurrence number of the most common value and second most common value of vector J, respectively), and $S2_f(J) = a$ such as $\#_a(J) = \#_{1st}(J)$. These experimental results mostly show the proportion of input vectors that satisfy $\mathscr{C}2_f^{[f]}$ and those that satisfy $\mathscr{C}2_f^{[f],s}$. These results have been obtained as follows. All the vectors in $\mathscr{V}_f^n$ have been generated for different values of n and of $\mathscr{V}$. First, the number of input vectors in $\mathscr{C}2_f^{[f]}$ has been enumerated. Then, for each input vector J not in $\mathscr{C}2_f^{[f]}$, we have made an inventory of all the input vectors $J1 \in \mathscr{V}_f^n$ such that $J1 \sim^f J$. Let $|J1 \sim^f J|$ be this number. We have then evaluated the ratio $r_J = \frac{|J1 \sim^f J| \wedge (S(J1) = S(J))}{|J1 \sim^f J|}$ to calculate the number of input vectors in $\mathscr{C}2_f^{[f],s,S(J)}$. We have $\mathscr{C}2_f^{[f],s,S(J)} = \{J' \in \mathscr{V}_f^n / \neg P^{[f]}(J') \wedge (r_{J'} \geq k) \wedge (S(J) = S(J'))\}$. Finally, the proportion of input vectors belonging to $\mathscr{C}2_f^{[f],s}$ is equal to $r = \frac{|\bigcup_{S(J) \in \mathscr{V}} \mathscr{C}2_f^{[f],s,S(J)}|}{|\mathscr{V}_f^n|}$. The lessons learned from these experimental results are consistent with the intuition. These results are shown in Fig-

ure 5. This figure depicts two graphs. The left graph of the figure plots the ratio of input vectors belonging to $\mathscr{C}2_f^{[f]}$ (y axis) against the number n of processes (x axis) for $|\mathscr{V}| = 2$ and 3. From this graph, we observe that this ratio strongly decreases with increasing values of n (this ratio decreases to 0.3% for $n = 9$ and $|\mathscr{V}| = 3$). This confirms the fact that favoring the value that appears the most often in an input vector despite up to f crashes and the presence of any other value is more difficult to attain when n increases. The right graph of the figure plots the ratio of input vectors belonging to $\mathscr{C}2_f^{[f],s}$ (y axis) against the number n of processes (x axis) for $|\mathscr{V}| = 2$ and 3. This ratio is obtained with $k = 0.7$. Contrary to the left one, both curves are less sensitive to increasing values of n, and the ratio remains above 30% of the total number of input vectors.

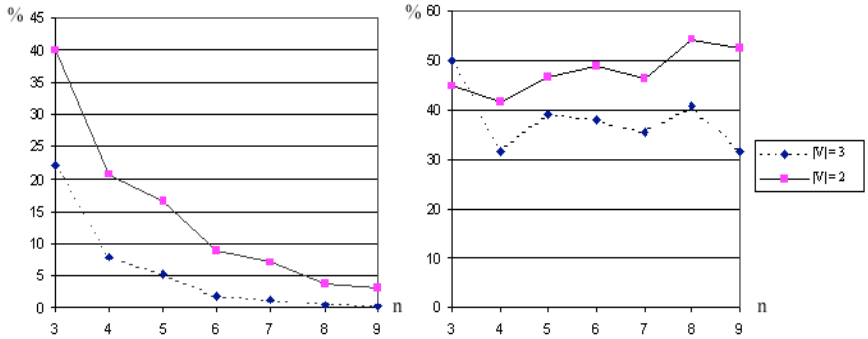

Figure 5: Ratio of Decision and Strong Convergence Vectors Against the Number of Processes

5. The Consensus Protocol

5.1. *Principles of the Protocol*

This section presents a distributed algorithm that solves the consensus problem described in Section 2.2. This algorithm implements the convergence-based approach. We assume a set of n processes connected

by stable links such that among them strictly less than $\lceil k_0 n \rceil = \frac{n}{2}$ processes may crash. Processes proceed in asynchronous and consecutive rounds. In contrast to several existing protocols, the protocol we propose is neither based on the rotating coordinator nor on the leader-based paradigms. Rather, all the processes have a symmetric role, and at each round any process is likely to determine the decision value. The protocol is described in Figure 6. A process p_i starts a consensus execution by invoking the function `Consensus(`v`)`, where v is the value p_i proposes. This function is made up of four concurrent tasks T_1, T_2, T_3, and T_4. The statement return(est_i) returns to p_i the decision value est_i which terminates p_i consensus execution.

Task T_1 is in charge of building p_i input vector V_i and querying the condition-based$^+$ module. Whenever p_i has received $\lceil k_0 n \rceil$ propose messages, and at each new receipt, p_i queries its condition-based$^+$ module with its local view V_i as parameter (lines 5-6).

Task T_2 is the main task of the protocol. At the beginning of a new round, p_i queries $UDTS_i$ to obtain its vector of time estimations $TimEst_i$ (line 10). As previously said in Section 2, the kth entry of $TimEst_i$ provides the time needed for p_i to receive $\lceil k_0 n \rceil + k$ propose messages. Process p_i first initializes an alarm clock with the time needed to receive $\lceil k_0 n \rceil$ propose messages, that is, $TimEst_i[0]$ (line 13). Then it broadcasts its proposed value to all the processes and enters a waiting loop. Process p_i exits this waiting loop either when it times out, or when its input vector V_i satisfies either $\mathscr{C}_f^{[f]}$ (*decide*=true) or $\mathscr{C}_f^{[f],s}$ (*strong*=true). During this waiting loop, p_i receives at least $\lceil k_0 n \rceil$ propose messages. Clearly, if the proposed values satisfy the f-acceptable condition (implemented within the condition-based$^+$ module), then p_i directly decides the value returned by this module, i.e., *est* (line 24). In that case, p_i exits from the wait until loop, reliably broadcasts the decision value to all the processes, and returns *est* (lines 16-18).

Task T_3 analyzes the result returned by the condition-based$^+$ module. First, p_i updates its estimate with $S(V_i)$ (line 23). Then it checks whether it is worth waiting for more estimates or not. This is implemented by testing whether *nbvalue* $\leq k + \lceil k_0 n \rceil - \#(\neg\perp)_{V_i}$ is true. If

this is true, this means that the receipt of *nbvalue* more estimates guarantees p_i either to decide (if V_i already belongs to a strong convergence area) or to converge toward a strong convergence area (if V_i belongs to a weak convergence area) (see Section 4.3). In both cases, p_i continues to wait in the wait until loop. If the above test is not satisfied, then p_i searches in $TimEst$, the smaller waiting time corresponding to the receipt of *nbvalue* estimates (line 26). If this waiting time exists (i.e., $UDTS_i$ is able to determine the time needed to receive *nbvalue* messages, that is at least *nbvalue* processes are still non-crashed), then p_i updates its alarm clock accordingly, and continues to wait in the waiting loop with this new delay. Otherwise, p_i cannot wait for any more estimates. Then it tests whether its input vector belongs to a strong convergence area. In the affirmative, it exits from the waiting loop, and enters the next round. Indeed, waiting for more estimates is not necessary since the distance of V_i to the closest decision vector is too large to be reached. Otherwise (lines 33 34), p_i strives to converge toward a strong convergence area. To this end, it updates its alarm clock with the time needed to receive the greatest number of estimates, and continues to wait in the waiting loop with this new delay.

Task T_4 is a decision task.

5.2. *Proof of the Consensus Protocol*

Lemma 5.1 *Infinitely often if 2Δ has elapsed then* $\Delta = Sup\{TimEst[i], i \in [1, n]\}$.

Proof The proof is by contradiction. Let V_i be the input vector of process p_i. Suppose that 2Δ time units have elapsed and $\Delta \neq Sup\{TimEst[i], i \in [1, n]\}$. Thus there exists some k such that $k = Min(e \in [0, \lceil k_0 n \rceil) | nbvalue \leq e + \lceil k_0 n \rceil - \#(\neg\perp)_{V_i})$ (line 26). From the property of $UDTS_i$, infinitely often Δ allows the receipt of at least $nbvalue$ messages before $t + 2\Delta$. By construction of the condition-based$^+$ module, V_i is such that *distance*$(V_i, C_f^{[f]}) = nbvalue$ if $V \in C_f^{[f],s}$, or *distance*$(V_i, C_f^{[f],s}) = nbvalue$ if $V \in C_f^{[f],w}$. At each new receipt of a propose message two cases are possible :

Function `Consensus`(ν)

(1) $r \leftarrow 0$; $est_i \leftarrow \nu$; $V_i[1..n] \leftarrow \perp$
decision $\leftarrow$ *false*; *Decide* $\leftarrow$ *false*; *Strong* $\leftarrow$ *false*;

(2) % `ReadTime()` : returns the local time system

Task T_1:
(3) **upon receipt of** `propose`(r, est_j)
(4) $V_i[j] \leftarrow est_j$;
(5) **if** $(\#(\neg\perp)_{V_i} \geq \lceil k_0 n \rceil)$ **then**
(6) `GetCond`$^+(V_i)$;
(7) **endif**
endTask

– case 1: the received propose message does not decrease $nbvalue$. Thus $nbvalue \leq k + \lceil k_0 n \rceil - (\#(\neg\perp)_{V_i} + 1)$, that is, $nbvalue \leq (k-1) + \lceil k_0 n \rceil - \#(\neg\perp)_{V_i}$. By definition of k, this is impossible.

– case 2: the received propose message decreases $nbvalue$. Thus, progressively, *nbvalue* decreases down to 0 before time $t + 2\Delta$ time units. Thus either *decide*=true or *strong*=true before $t + 2\Delta$ time units have elapsed, which contradicts the assumption of the lemma.

This ends the proof of the lemma. $\square_{Lemma\ 5.1}$

Lemma 5.2 *Infinitely often at the end of a round r, all non crashed processes have the same estimate.*

Proof Let us consider any two processes p_i and p_j. Several cases are considered.

– p_i decides. In this case, three sub-cases are possible:

- p_j decides : Due to the acceptability of the condition, we have $S(V_i) = S(V_j)$ leading by construction of the algorithm to $est_i = est_j$.

- p_j strongly converges : Due to the Convergence Agreement property we have $S(V_i) = S(V_j)$. Thus by construction of the algorithm, we have $est_i = est_j$.

- p_j weakly converges : By construction of the protocol, 2Δ time units have elapsed. From the property of $UDTS_j$, infinitely often Δ time units allow p_j to receive all the messages sent to it. Thus p_j receives at least all the estimates p_i has received. Thus by function S, $S(V_i) = S(V_j)$.

– No process decides and all the processes exit the wait until loop with their input vector satisfying a strong convergence area. By the strong convergence agreement property all the processes p_i and p_j satisfy $S(V_i) = S(V_j)$, and thus have the same estimate at the end of the round.

– 2Δ time units have elapsed for all the non-crashed processes. Then infinitely often all the processes have received all the messages sent to them (by lemma 5.1). Thus, infinitely often all the processes build the same local view and by function S, obtain the same estimates.

– No process decides, some processes exit the waiting loop after $t + 2\Delta$ time units, and the other ones belong to a strong convergence area. Then all the processes that exit the waiting loop upon time-out have received all the messages sent to them. By construction of the protocol, either they strongly converge or they decide. Thus, by the strong convergence agreement area, they all complete the round with the same estimate.

$\square_{Lemma\ 5.2}$

Lemma 5.3 *If a correct p_i does not begin the round $r + 1$ after after having executed round r then p_i decides at the end of the round r.*

Proof The proof is by contradiction. Suppose that p_i does not decide in round r and does not begin round $r + 1$. It must be the case that p_i is blocked forever in the waiting loop, that is *decide*=false and *strong*=false. By assumption of the model, p_i receives at least $n - f$ messages. Thus eventually, p_i time-outs (after 2Δ time units), exits the wait until loop, and begins round $r+1$. This contradicts the assumption, and concludes the lemma. $\square_{Lemma\ 5.3}$

Lemma 5.4 *Eventually, at least one process decides.*

Proof Let r be the current round. If some correct process p_i does not begin round r then p_i must have previously decided by lemma 5.3. Suppose that all the non-crashed processes begin round r. Let p_i be such a process. If V_i satisfies $\mathscr{C}_f^{[f]}$, then p_i decides. Otherwise, due to lemma 5.2 infinitely often all the processes complete round r with the same estimate est. Thus, in round $r+1$, all the local views are comparable to an input vector J=[est,....est] with $J \in \mathscr{C}^{[f]}$. Thus, each local view satisfies $C_f^{[f]}$ and hence decides. This ends the proof of the lemma. $\square_{Lemma\ 5.4}$

Lemma 5.5 *If some process decides then all non crashed processes decide.*

Proof Straightforward from Task T_4. $\square_{Lemma\ 5.5}$

Lemma 5.6 *Eventually, all correct processes decide.*

Proof Straightforward from lemmas 5.4 and 5.5. $\square_{Lemma\ 5.6}$

Lemma 5.7 *Infinitely often, all the correct processes decide in a bounded number of communication steps.*

Proof Let r be a round such that all non-crashed processes have executed r. By lemma 5.2, infinitely often all the non-crashed processes complete round r with the same estimate, and if no decision is reached, execute round $r+1$ with a view in $C_f^{[f]}$. Thus, a decision is performed infinitely often in 2 communication steps. $\square_{Lemma\ 5.7}$

6. Conclusion

In this paper, we have presented an extension of the condition-based approach introduced by Mostefaoui et al in 2001. The condition-based approach is related to the consensus impossibility result shown by Fisher Lynch and Paterson result in 1995 (Fisher *et al.,* 1985). Briefly, the condition-based approach finds restriction on the set of input vectors proposed by the set of processes for which the consensus problem is directly solvable (i.e., in one communication step). We have extended this condition to significantly augment the number of input vectors for which the consensus problem becomes solvable. To this end, we have characterized, for a given acceptable condition, the set of input vectors that makes all the processes converge, with a computable probability, toward the same decision. Our approach has been applied to a distributed algorithm to solve the consensus problem: by adopting the convergence value as new input value, the probability to converge toward a good patterned one (i.e., a vector that satisfies an acceptable condition) increases to be equal to 1. For example, experimental results show that favoring the value that appears the most often in an input vector despite up to f crashes and the presence of any other value is not sensitive to increasing values of n, which confirms the interest of our approach.

7. References

Anceaume E., Mourgaya E., "Unreliable Distributed Timing Scrutinizer: Adapting Asynchronous Algorithms to the Environment" , *Proc. of the 5th IEEE International Symposium on Object-Oriented Real-time distributed Computing (ISORC 2002)*, 2002.

Ben-Or M., "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols" , *2nd ACM Symp. on Principles of Distributed Computing (PODC'83)*, p. 27-30, 1983.

Chandra T., Toueg S., "Unreliable Failure Detectors for Reliable Distributed Systems" , *Journal of the ACM*, vol. 43,2, p. 225-267, 1996.

Chaudhuri S., "More *Choices* Allow More *faults*: Set Consensus Problems in Totally Asynchronous Systems" , *Information and Computation*,105, p. 132-158, 1993.

Cristian F., Fetzer C., "The Timed Asynchronous Distributed System Model" , *IEEE,Transactions on parallel and Distributed Systems*, p. 642-657, 1999.

Dolev D., Lynch N., Pinter S., Stark E., Wheihl W., "Reaching Approximate Agreement in the Presence of Faults" , *Journal of ACM*, vol. 33,3, p. 499-516, 1986.

Fisher M., Lynch N., Paterson M., "Impossibility of Distributed Consensus with one Faulty Process" , *Journal of the ACM*, vol. 32,2, p. 374-382, 1985.

Mostéfaoui A., Mourgaya E., RaÔpin Parvédy P., Raynal M., "Evaluating the Condition-Based Approach to Solve Consensus" , *Proc. of the 2003 International Conference on Dependable Systems and Networks (DSN-2003)*, IEEE, 2003.

Mostéfaoui A., Rajsbaum S., Raynal M., "Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems" , *Proc. of the 33rd ACM Symposium on Theory of Computing (STOC'01)*, ACM Press, p. 153-162, 2001a.

Mostéfaoui A., Rajsbaum S., Raynal M., "A Versatile and Modular Consensus Protocol" , *Proc. of the 2002 International Conference on Dependable Systems and Networks (DSN-2002)*, IEEE, 2002.

Mostéfaoui A., Rajsbaum S., Raynal M., Roy M., "Efficient Condition-based Consensus" , *Proc. of the 8th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO'01)*, Carleton Univ. Press, p. 275-291, 2001b.

Mostéfaoui A., Rajsbaum S., Raynal M., Roy M., "A Hierarchy of Conditions for Consensus Solvability" , *Proc. of the 20th ACM Symposium on Principles of Distributed Computing (PODC-2001)*, ACM Press, Newport, RI, 2001c.

Shostak R., Pease M., Lamport L., "Reaching Agreement in Presence of Faults" , *Journal of the ACM*, 1981.

Verissimo P., Raynal M., *Recent Advances in Distributed Systems*, S. Krakowiak and S. Shrivastava, Springer-Verlag, chapter Time, Clock and Temporal Order, 2000.

Task T_2:

```
(8)  while (¬decision) do
(9)      r ← r + 1; k ← 0;
(10)     TimEst_i ← Get∆([⌈k_0 n⌉, n]); % p_i queries UDTS_i
                                         % to get the vector of time estimations
(11)     Broadcast(propose(r, est_i));
(12)     t=ReadTime();
(13)     ∆ ← TimEst_i[k];
(14)     wait until ((ReadTime()=t + 2∆) ∧
                 (receipt of at least ⌈k_0 n⌉ propose(r,-)) ∨ Decide ∨ Strong)
(15)     est_i ← estimate_i;
(16)     if (Decide) then
(17)         decision ← true;
(18)         Reliable_Broadcast(decide(est_i));
(19)         return(est_i);
(20)     endif
(21) endwhile
endTask
```

Task T_3**:**

```
(22) upon receipt of PutCond+(est,nbvalue,decide,strong)
(23)     estimate_i ← est;
(24)     Decide ← decide;
(25)     if (nbvalue > k + ⌈k_0 n⌉ − #(¬⊥)_{V_i}) then
(26)         k ← Min(Min(e ∈ [0, f]/nbvalue ≤ e + ⌈k_0 n⌉ − #(¬⊥)_{V_i}), n);
(27)         if TimEst_i[k] ≠ ⊥ then
(28)             ∆ ← TimEst_i[k];
(29)         else
(30)             if strong then
(31)                 Strong ← true
(32)             else
(33)                 k ← Sup{e ∈ [0, f]/ TimEst_i[e] };
(34)                 ∆ ← TimEst_i[k];
(35)             endif
(36)         endif
(37)     else
(38)         skip ;
(39)     endif
endTask
```

Task T_4**:**
(40) **upon receipt of** `decide(`est`)` $\wedge \neg$*decision*
(41) *decision* $\leftarrow$ *true*;
(42) return(est);
endTask

Figure 6: Consensus Protocol Run by Process p_i