From an Intermittent Rotating Star to a Leader

Antonio Fernández Anta[1] and Michel Raynal[2]

[1] LADyR, GSyC, Universidad Rey Juan Carlos, 28933 Móstoles, Spain
anto@gsyc.escet.urjc.es
[2] IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, France
raynal@irisa.fr

Abstract. Considering an asynchronous system made up of n processes and where up to t of them can crash, finding the weakest assumptions that such a system has to satisfy for a common leader to be eventually elected is one of the holy grail quests of fault-tolerant asynchronous computing. This paper is a step in such a quest. It has two main contributions. First, it proposes an asynchronous system model, in which an eventual leader can be elected, that is weaker and more general than previous models. This model is captured by the notion of *intermittent rotating t-star*. An *x-star* is a set of $x+1$ processes: a process p (the center of the star) plus a set of x processes (the points of the star). Intuitively, assuming logical times rn (round numbers), the *intermittent rotating t-star* assumption means that there are a process p, a subset of the round numbers rn, and associated sets $Q(rn)$ such that each set $\{p\} \cup Q(rn)$ is a t-star centered at p, and each process of $Q(rn)$ receives from p a message tagged rn in a timely manner or among the first $(n-t)$ messages tagged rn it ever receives. The star is called *t-rotating* because the set $Q(rn)$ is allowed to change with rn. It is called *intermittent* because the star can disappear during finite periods. This assumption, not only combines, but generalizes several synchrony and time-free assumptions that have been previously proposed to elect an eventual leader (e.g., eventual t-source, eventual t-moving source, message pattern assumption). Each of these assumptions appears as a particular case of the *intermittent rotating t-star* assumption. The second contribution of the paper is an algorithm that eventually elects a common leader in any system that satisfies the *intermittent rotating t-star* assumption. That algorithm enjoys, among others, two noteworthy properties. Firstly, from a design point of view, it is simple. Secondly, from a cost point of view, only the round numbers can increase without bound. This means that, be the execution finite or infinite, be links timely or not (or have the corresponding sender crashed or not), all the other local variables (including the timers) and message fields have a finite domain.

Keywords: Assumption coverage, Asynchronous system, Distributed algorithm, Eventual t-source, Eventual leader, Failure detector, Fault-tolerance, Message pattern, Moving source, Omega, Partial synchrony, Process crash, System model, Timely link.

1 Introduction

1.1 Leader Oracle: Motivation

A failure detector is a device (also called oracle) that provides the processes with guesses on which processes have failed (or not failed) [3,21]. According to the

E. Tovar, P. Tsigas, and H. Fouchal (Eds.): OPODIS 2007, LNCS 4878, pp. 189–203, 2007.

properties associated with these estimates, several failure detector classes can be defined. It appears that failure detector oracles are at the core of a lot of fault-tolerant protocols encountered in asynchronous distributed systems. Among them, the class of *leader* failure detectors is one of the most important. This class, also called the class of leader oracles, is usually denoted Ω. (When clear from the context, the notation Ω will be used to denote either the oracle/failure detector class or an oracle of that class.) Ω provides the processes with a *leader* primitive that outputs a process id each time it is called, and such that, after some finite but unknown time, all its invocations return the same id, that is the identity of a correct process (a process that does not commit failures). Such an oracle is very weak: (1) a correct leader is eventually elected, but there is no knowledge on when it is elected; (2) several (correct or not) leaders can co-exist before a single correct leader is elected.

The oracle class Ω has two fundamental features. The first lies on the fact that, despite its very weak definition, it is powerful enough to allow solutions to fundamental problems such as the consensus problem [4]. More precisely, it has been shown to be the weakest class of failure detectors that allows consensus to be solved in message-passing asynchronous systems with a majority of correct processes (let us remind that, while consensus can be solved in synchronous systems despite Byzantine failures of less than one third of the processes [14], it cannot be solved in asynchronous distributed systems prone to even a single process crash [7]). Basically, an Ω-based consensus algorithm uses the eventual leader to impose a value to all the processes, thereby providing the algorithm liveness. Leader-based consensus protocols can be found in [9,13,18]. The second noteworthy feature of Ω lies on the fact that it allows the design of *indulgent* protocols [8]. Let P be an oracle-based protocol that produces outputs, and PS be the safety property satisfied by its outputs. P is *indulgent with respect to its underlying oracle* if, whatever the behavior of the oracle, its outputs never violate the safety property PS. This means that each time P produces outputs, they are correct. Moreover, P always produces outputs when the underlying oracle meets its specification. The only case where P can be prevented from producing outputs is when the implementation of the underlying oracle does not meet its specification. (Let us notice that it is still possible that P produces outputs despite the fact that its underlying oracle does not work correctly.) Interestingly, Ω is a class of oracles that allows designing indulgent protocols [8,9]. More precisely, due to the very nature of an *eventual* leader, it cannot be known in advance when that leader is elected; consequently, the main work of an Ω-based consensus algorithm is to keep its safety property, i.e., guarantee that no two different values can be decided before the eventual leader is elected.

Unfortunately, Ω cannot be implemented in pure asynchronous distributed systems where processes can crash. (Such an implementation would contradict the impossibility of solving consensus in such systems [7]. Direct proofs of the impossibility to implement Ω in pure crash-prone asynchronous systems can be found in [2,19].) But thanks to indulgence, this is not totally bad news. More precisely, as Ω makes it possible the design of indulgent protocols, it is interesting to design "approximate" protocols that do their best to implement Ω on top of the asynchronous system itself. The periods during which their best effort succeeds in producing a correct implementation of the oracle (i.e., there is a single leader and it is alive) are called "good" periods (and then, the upper layer Ω-based protocol produces outputs and those are correct). During the other periods (sometimes called "bad" periods, e.g., there are several leaders or the leader is a crashed process), the upper layer Ω-based protocol never produces erroneous outputs. The only bad thing that can then happen is that this protocol can be prevented from producing outputs, but when a new long enough good period appears, the upper layer Ω-based protocol can benefit from that period to produce an output.

A main challenge of asynchronous fault-tolerant distributed computing is consequently to identify properties that are at the same time "weak enough" in order to be satisfied "nearly always" by the underlying asynchronous system, while being "strong enough" to allow Ω to be implemented during the "long periods" in which they are satisfied.

1.2 Existing Approaches to Implement Ω

Up to now, two main approaches have been investigated to implement Ω in crash-prone asynchronous distributed systems. Both approaches enrich the asynchronous system with additional assumptions that, when satisfied, allow implementing Ω. These approaches are orthogonal: one is related to timing assumptions, the other is related to message pattern assumptions.

The eventual timely link approach. The first approach considers that the asynchronous system eventually satisfies additional *synchrony* properties. Considering a reliable communication network, the very first papers (e.g., [15]) assumed that all the links are *eventually timely*[1]. This assumption means that there is a time τ_0 after which there is a bound δ -possibly unknown- such that, for any time $\tau \geq \tau_0$, a message sent at time τ is received by time $\tau + \delta$.

This approach has then been refined to obtain weaker and weaker assumptions. It has been shown in [1] that it is possible to implement Ω in a system where communication links are unidirectional, asynchronous, and lossy, provided that there is a correct process whose $n-1$ output links are eventually timely (n being the total number of processes). This assumption has further been weakened in [2] where it is shown that Ω can be built as soon as there is a correct process that has only t eventually timely links (where t is a known upper bound on the number of processes that can crash); such a process is called an *eventual t-source*. (Let us notice that, after the receiver has crashed, the link from a correct process to a crashed process is always timely.)

Another time-based assumption has been proposed in [16] where the notion of *eventual t-accessibility* is introduced. A process p is eventual t-accessible if there is a time τ_0 such that, at any time $\tau \geq \tau_0$, there is a set $Q(\tau)$ of t processes such that $p \notin Q(\tau)$ and a message broadcast by p at τ receives a response from each process of $Q(\tau)$ by time $\tau + \delta$ (where δ is a bound known by the processes). The very important point here is that the set $Q(\tau)$ of processes whose responses have to be received in a timely manner is not fixed and can be different at distinct times.

The notions of eventual t-source and eventual t-accessibility cannot be compared (which means that none of them can be simulated from the other). In a very interesting

[1] Actually, the Ω protocol presented in [15] only requires that the output links of the correct process with the smallest identity to be eventually timely.

way these two notions have been combined in [11] where is defined the notion of *eventual t-moving source*. A process p is an eventual t-moving source if there is a time τ_0 such that at any time $\tau \geq \tau_0$ there is a set $Q(\tau)$ of t processes such that $p \notin Q(\tau)$ and a message broadcast by p at τ is received by each process in $Q(\tau)$ by time $\tau + \delta$. As we can see, the *eventual t-moving source* assumption is weaker than the *eventual t-source* as the set $Q(\tau)$ can vary with τ.

Other time-based approaches are investigated in [5,12]. They consider weak assumptions on both the initial knowledge of processes and the network behavior. Protocols building Ω are presented [5,12] that assume the initial knowledge of each process is limited to its identity and the fact that identities are totally ordered (so, a process knows neither n nor t). An unreliable broadcast primitive allows the processes to communicate. One of the protocols presented in [5] is communication-efficient (after some time a single process has to send messages forever) while, as far as the network behavior is concerned, it only requires that each pair of correct processes be connected by fair lossy links, and there is a correct process whose output links to the rest of correct processes are eventually timely. It is shown in [12] that Ω can be built as long as there is one correct process that can reach the rest of the correct processes via eventually timely paths.

The message pattern approach. A totally different approach to build Ω has been introduced in [17]. That approach does not rely on timing assumptions and timeouts. It states a property on the *message exchange pattern* that, when satisfied, allows Ω to be implemented. The statement of such a property involves the system parameters n and t.

Let us assume that each process regularly broadcasts queries and, for each query, waits for the corresponding responses. Given a query, a response that belongs to the first $(n - t)$ responses to that query is said to be a *winning* response. Otherwise, the response is a *losing* response (then, that response is slow, lost or has never been sent because its sender has crashed). It is shown in [19] that Ω can be built as soon as the following behavioral property is satisfied: "There are a correct process p and a set Q of t processes such that $p \notin Q$ and eventually the response of p to each query issued by any $q \in Q$ is always a winning response (until -possibly- the crash of q)." When $t = 1$, this property becomes: "There is a link connecting two processes that is never the slowest (in terms of transfer delay) among all the links connecting these two processes to the rest of the system." A probabilistic analysis for the case $t = 1$ shows that such a behavioral property on the message exchange pattern is practically always satisfied [17].

This *message pattern* approach and the *eventual timely link* approaches cannot be compared. Interestingly, the message pattern approach and the eventual t-source approach have been combined in [20]. This combination shows that Ω can be implemented as soon as there is a correct process p and a time τ_0 after which there is a set Q of t processes q such that $p \notin Q$ and either (1) each time a process $q \in Q$ broadcasts a query, it receives a winning response from p, or (2) the link from p to q is timely. As it can be seen, if only (1) is satisfied, we obtain the *message pattern* assumption, while, if only (2) is satisfied, we obtain the *eventual t-source* assumption. More generally, here, the important fact is that the message pattern assumption and the timely link assumption are combined at the "finest possible" granularity level, namely, the link level.

1.3 Content of the Paper: Towards Weaker and Weaker Synchrony Assumptions

A quest for a fault-tolerant distributed computing holy grail is looking for the *weakest synchrony assumptions* that allow implementing Ω. Differently from the quest for the weakest information on failures that allows solving the consensus problem (whose result was Ω [4]), it is possible that this quest be endless. This is because we can envisage lots of base asynchronous computation models, and enrich each of them with appropriate assumptions that allow implementing Ω in the corresponding system. Such a quest should be based on a well-formalized definition of a low level asynchronous model, including all the models in which Ω can be implemented. There is no guarantee that such a common base model exists.

So, this paper is only a step in that direction. It considers the classical asynchronous computing model where processes can crash. They communicate through a reliable network [3,7]. (Fair lossy links could be used instead of reliable links[2] but we do not consider that possibility in order to keep the presentation simple.) The paper shows that it is possible to implement Ω in an asynchronous system from a synchrony assumption weaker than any of the previous ones, namely, *eventual* t*-source*, *eventual* t*-moving source*, or the *message pattern* assumption. Interestingly, these specific assumptions become particular cases of the more general (and weaker) assumption that is proposed. In that sense, the paper not only proposes a weaker assumption, but has also a generic dimension.

The proposed behavioral assumption (that we denote $\mathcal{A}$) requires that each process regularly broadcasts ALIVE(rn) messages, where rn is an increasing round number (this can always be done in an asynchronous system). The sending of ALIVE(rn) messages by the processes can be seen as an *asynchronous round*, each round number defining a new round.

To make easier the presentation we describe first an assumption $\mathcal{A}^+$ of which $\mathcal{A}$ is a weakening. $\mathcal{A}^+$ is as follows. There is a correct process p and a round number RN_0 such that, for each $rn \geq RN_0$, there is a set $Q(rn)$ of t processes such that $p \notin Q(rn)$ and for each process $q \in Q(rn)$ either (1) q has crashed, or (2) the message ALIVE(rn) sent by p is received by q at most δ time units after it has been sent (the corresponding bound δ can be unknown), or (3) the message ALIVE(rn) sent by p is received by q among the first $(n-t)$ ALIVE(rn) messages received by q (i.e., it is a winning message among ALIVE(rn) messages received by q). It is easy to see, that if only (1) and (2) are satisfied, $\mathcal{A}^+$ boils down to the eventual t-moving source assumption, while if only (1) and (3) are satisfied, it boils down to a *moving* version of the message pattern assumption (because the set $Q()$ can change over time). The set of processes $\{p\} \cup Q(rn)$ defines a star centered at p. As it must have at least t points (links), we say it is a t*-star*. Moreover, as $Q(rn)$ can change at each round number, we say that p is the *center* of an *eventual*

---

[2] This can easily be done by using message acknowledgments and piggybacking: a message is piggybacked on the next messages until it has been acknowledged. So, a message sent by the underlying communication protocol can be made up of several messages sent by the upper layer algorithm. It is nevertheless important to remark that such a piggybacking + acknowledgment technique is viable only if the size of the messages sent by the underlying communication protocol remains manageable.

rotating t-star ("eventual" because there is an arbitrary finite number of round numbers during which the requirement may not be satisfied).

While $\mathcal{A}^+$ allows implementing Ω, it appears that a weakened form of that assumption is sufficient. This is the assumption $\mathcal{A}$. It is sufficient that p be the center of an eventual rotating t-star only for a subset of the round numbers. More precisely, $\mathcal{A}$ requires that there is an infinite sequence $S = s_1, s_2, \ldots$ of (not necessarily consecutive) round numbers, and a bound D (not necessarily known), such that, $\forall k \geq 1$, $s_{k+1} - s_k \leq D$, and there is a process p that is the center of a rotating t-star when we consider only the round numbers in S. We call such a configuration an *eventual intermittent rotating t-star* (in fact, the "eventual" attribute could also be seen as being part of the "intermittent" attribute).

Basically, the difference between $\mathcal{A}^+$ and $\mathcal{A}$ is related to the notion of observation level [10]. While $\mathcal{A}^+$ considers a base level including all the round numbers, $\mathcal{A}$ provides an abstraction level (the sequence S) that eliminates the irrelevant round numbers. Of course, as it is not known in advance which are the relevant round numbers (i.e., S), an $\mathcal{A}$-based algorithm has to consider a priori all the round numbers and then find a way to dynamically skip the irrelevant ones.

After having introduced $\mathcal{A}^+$ and $\mathcal{A}$, the paper presents, in an incremental way, an $\mathcal{A}^+$-based algorithm and an $\mathcal{A}$-based algorithm that build a failure detector oracle of the class Ω. The $\mathcal{A}$-based algorithm enjoys a noteworthy property, namely, in an infinite execution, only the round numbers increase forever. All the other local variables and message fields remain finite. This means that, among the other variables, all the timeout values (be the corresponding link eventually timely or not) eventually stabilize. From an algorithmic mechanism point of view, the proposed algorithm combines new ideas with mechanisms also used in [2,5,11,17,20].

All the proofs and additional technical developments can be found in [6].

2 Definitions

2.1 Basic Distributed System Model

We consider a system formed by a finite set Π of $n \geq 2$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. Process identifiers are totally ordered. Without loss of generality we assume that $ID(p_i) < ID(p_j)$, when $i < j$, and use $ID(p_i) = i$. We sometimes use p and q to denote processes. A process executes steps (a step is the reception of a set of messages with a local state change, or the sending of messages with a local state change). 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 a process that is not correct. As previously indicated, t denotes the maximum number of processes that can crash ($1 \leq t < n$).

Processes communicate and synchronize by sending and receiving messages through links. Every pair of processes (p, q) is connected by two directed links, denoted $p \rightarrow q$ and $q \rightarrow p$. Links are assumed to be reliable: they do not create, alter, or lose messages. In particular, if p sends a message to q, then eventually q receives that message unless one of them fails. There is no assumption about message transfer delays (moreover, the links are not required to be FIFO).

Processes are synchronous in the sense that there are lower and upper bounds on the number of processing steps they can execute per time unit. Each process has also a local clock that can accurately measure time intervals. The clocks of the processes are not synchronized. To simplify the presentation, and without loss of generality, we assume in the following that the execution of the local statements take no time. Only the message transfers consume time.

In the following, $AS_{n,t}[\emptyset]$ denotes an asynchronous distributed system as just described, made up of n processes among which up to $t < n$ can crash. More generally, $AS_{n,t}[P]$ will denote an asynchronous system made up of n processes among which up to $t < n$ can crash, and satisfying the additional assumption P (so, $P = \emptyset$ means that the system is a *pure* asynchronous system).

We assume the existence of a global discrete clock. This clock is a fictional device which is not known by the processes; it is only used to state specifications or prove protocol properties. The range of clock values is the set of real numbers.

2.2 The Oracle Class Ω

Ω has been defined informally in the introduction. A leader oracle is a distributed entity that provides the processes with a function leader() that returns a process id each time it is invoked. A unique correct process is eventually elected but there is no knowledge of when the leader is elected. Several leaders can coexist during an arbitrarily long period of time, and there is no way for the processes to learn when this "anarchy" period is over. A leader oracle satisfies the following property [4]:

– Eventual Leadership: There is a time τ and a correct process p such that any invocation of leader() issued after τ returns the id of p.

Ω-based consensus algorithms are described in [9,13,18] for asynchronous systems where a majority of processes are correct ($t < n/2$). These algorithms can then be used as a subroutine to solve other problems such as atomic broadcast (e.g., [3,13]).

As noticed in the introduction, whatever the value of $t \in [1, n-1]$, Ω cannot be implemented in $AS_{n,t}[\emptyset]$. Direct proofs of this impossibility can be found in [2,19] ("direct proofs" means that they are not based on the impossibility of asynchronously solving a given problem such as the consensus problem [7]).

3 The Additional Assumption $\mathcal{A}$

This section defines a system model, denoted $AS_{n,t}[\mathcal{A}]$ ($AS_{n,t}[\emptyset]$ enriched with the assumption $\mathcal{A}$) in which failure detectors of the class Ω can be built. (Said differently, this means that Ω can be implemented in all the runs of $AS_{n,t}[\emptyset]$ that satisfy $\mathcal{A}$.)

Process behavior requirement. The assumption $\mathcal{A}$ requires that each process p_i *regularly* broadcasts ALIVE(rn) messages (until it possibly crashes). The parameter rn is a round number that, for each process p_i, takes the successive values $1, 2, \ldots$

Let $send\_time(i, rn)$ be the time at which p_i broadcasts ALIVE(rn). The words "regularly broadcasts" means that the duration separating two broadcasts by the same process is bounded. More formally, there is a bound β (not necessarily known by the processes) such that, for any round number rn and any process p_i (until it possibly crashes), we have $0 < send\_time(i, rn+1) - send\_time(i, rn) \leq \beta$. It is important to notice that, given two different processes, there is no relation linking $send\_time(i, rn)$ and $send\_time(j, rn)$. It is easy to see that this broadcast mechanism can be implemented in $AS_{n,t}[\emptyset]$.

In the text of the algorithms, "**repeat regularly** ST" means that two consecutive executions of the statement ST are separated by at most β time units.

Definitions. According to the time or the order in which it is received, an ALIVE(rn) message can be δ-*timely* or *winning*. These notions are central to state the assumptions $\mathcal{A}^+$ and $\mathcal{A}$. It is important to remark that they are associated with messages, not with links. Let δ denote a bounded value.

Definition 1. *A message* ALIVE(rn) *is* δ-timely *if it is received by its destination process at most δ time units after it has been sent.*

Definition 2. *A message* ALIVE(rn) *is* winning *if it belongs to the first* $(n-t)$ ALIVE (rn) *messages received by its destination process.*

System model $AS_{n,t}[\mathcal{A}^+]$. The additional assumption $\mathcal{A}^+$ is the following: There is a correct process p, a bound δ, and a finite round number RN_0, such that for any $rn \geq RN_0$, there is a set of processes $Q(rn)$ satisfying the following properties:

- A1: $p \notin Q(rn)$ and $|Q(rn)| \geq t$ (i.e., $\{p\} \cup Q(rn)$ is a t-star centered at p), and
- A2: For any $q \in Q(rn)$ (i.e., any point of the star), one of the following properties is satisfied: (1) q has crashed, or (2) the message ALIVE(rn) is δ-timely, or (3) the message ALIVE(rn) is winning.

It is important to see that p, δ, and RN_0 are not known in advance, and can never be explicitly known by the processes. As said in the introduction, the process p that satisfies $\mathcal{A}^+$ is the center of an eventual rotating t-star.

$\mathcal{A}^+$ includes several dynamicity notions. One is related to the fact that the sets $Q()$ are not required to be the same set, i.e., $Q(rn_1)$ and $Q(rn_2)$ can be different for $rn_1 \neq rn_2$. This is the *rotating* notion (first introduced in [11,16] under the name *moving* set). A second dynamicity notion is the fact that two points of the star $\{p\} \cup Q(rn)$ (e.g., $p \rightarrow q1$ and $p \rightarrow q2$), are allowed to satisfy different properties, one satisfying the "δ-timely" property, while the other satisfying the "winning" property. Finally, if the point q appears in $Q(rn_1)$ and $Q(rn_2)$ with $rn_1 \neq rn_2$, it can satisfy the "δ-timely" property in $Q(rn_1)$ and the "winning" property in $Q(rn_2)$.

It is important to notice that the assumption $\mathcal{A}^+$ places constraints only on the messages tagged ALIVE. This means that, if an algorithm uses messages tagged ALIVE plus messages with other tags, there is no constraint on the other messages, even if they use the same links as the ALIVE messages.

System model $AS_{n,t}[\mathcal{A}]$. As indicated in the introduction, $\mathcal{A}$ is a weakening of $\mathcal{A}^+$ that allows the previous properties to be satisfied by only a subset of the round numbers. (None of the previous assumptions proposed so far have investigated such an assumption weakening.)

The additional assumption $\mathcal{A}$ is the following: There is a correct process p, a bound δ, a bound D, and a finite round number RN_0, such that:

- There is an infinite sequence S of increasing round numbers $s_1 = RN_0, s_2, \ldots, s_k, s_{k+1}, \ldots$, such that $s_{k+1} - s_k \leq D$, (so, the round numbers in S are not necessarily consecutive), and
- For any $s_k \in S$ there is a set of processes $Q(s_k)$ satisfying the properties A1 and A2 previously stated.

When $D = 1$, $\mathcal{A}$ boils down to $\mathcal{A}^+$. So, $\mathcal{A}$ weakens $\mathcal{A}^+$ by adding another dynamicity dimension, namely, a dimension related to time. It is sufficient that the rotating t-star centered at p appears from time to time in order Ω can be built. This is why we say that $\mathcal{A}$ defines an *intermittent rotating* t*-star*. The limit imposed by $\mathcal{A}$ to this dynamicity dimension is expressed by the bound D.

4 An $\mathcal{A}^+$-Based Leader Algorithm

This section presents and proves an algorithm that builds a failure detector of the class Ω in $AS_{n,t}[\mathcal{A}^+]$. This algorithm will be improved in the next sections to work in $AS_{n,t}[\mathcal{A}]$ (Section 5), and then to have only bounded variables (Section 6).

4.1 Principles and Description of the Algorithm

The algorithm is based on the following idea (used in one way or another in several leader protocols -e.g., [2,17]-): among all the processes, a process p_i elects as its current leader the process it suspects the least to have crashed (if several processes are the least suspected, p_i uses their ids to decide among them).

Local variables. To attain this goal each process p_i uses the following local variables:

- s_rn_i and r_rn_i are two round number variables. s_rn_i is used to associate a round number with each ALIVE() message sent by p_i. When $s\_rn_i = a$, p_i has executed up to its ath sending round.
  r_rn_i is the round number for which p_i is currently waiting for ALIVE() messages. When $r\_rn_i = b$, p_i is currently executing its bth receiving round.
  Sending rounds and receiving rounds are not synchronized (separate tasks are associated with them).
- $timer_i$ is p_i's local timer.
- $susp\_level_i[1..n]$ is an array such that $susp\_level_i[j]$ counts, from p_i's point of view, the number of rounds during which p_j has been suspected to have crashed by at least $(n - t)$ processes.
- $rec\_from_i[1..]$ is an array such that $rec\_from_i[rn]$ keeps the ids of the processes from which p_i has received an ALIVE(rn) message while $rn \geq r\_rn_i$ (if $rn < r\_rn_i$ when the message arrives, then it is too late and is consequently discarded).

– $suspicions_i[1.., 1..n]$ is an array such that $suspicions_i[rn, j]$ counts, as far as the receiving round rn is concerned, how many processes suspect that p_j has crashed.

Process behavior. The algorithm for a process p_i is described in Figure 1. It is made up of two tasks. The task $T1$ (Lines 1-3) is the sending task. In addition to its round number, each ALIVE() message carries the current value of the array $susp\_level_i$ (this gossiping is to allow the processes to converge on the same values for those entries of the array that stop increasing).

The task $T2$ is the main task. When leader() is locally invoked, it returns the id of the process that locally is the least suspected (Lines 19-21). If several processes are the least suspected, their ids are used to decide among them[3]. When an ALIVE(rn, sl) message is received, $T2$ updates accordingly the array $susp\_level_i$, and $rec\_from_i[rn]$ if that message is not late (i.e., if $r\_rn_i \geq rn$). The core of the task $T2$ is made up of the other two sets of statements.

– Lines 8-12. The timer $timer_i$ is used to benefit from the "δ-timely message" side of the assumption $\mathcal{A}^+$, while the set $rec\_from_i[r\_rn_i]$ is used to benefit from its "winning message" side. At each receiving phase r_rn_i, p_i waits until both the timer has expired and it has received $(n - t)$ ALIVE$(rn, *)$ messages with $rn = r\_rn_i$.

  When this occurs, as far as the receiving phase r_rn_i is concerned, p_i suspects all the processes p_k from which it has not yet received ALIVE$(r_rn_i, *)$ message. It consequently informs all the processes about these suspicions (associated with the receiving phase r_rn_i) by sending to all a SUSPICION$(r_rn_i, suspects)$ message (Line 10). Then, p_i proceeds to the next receiving phase (Line 12). It also resets the timer for this new (r_rn_ith) waiting phase (Line 11).

  The timer has to be reset to a value higher than the previous one when p_i discovers that it has falsely suspected some processes because its timer expired too early[4]. A way to ensure that the timeout value increases when there are such false suspicions, is adopting a conservative approach, namely, systematically increasing the timeout value. So, a correct statement to reset the timer (at Line 11) could be "**set** $timer_i$ **to** s_rn_i" (or to r_rn_i) as these round numbers monotonically increase.

  It appears (see the proof) that $susp\_level_i[j]$ is unbounded if p_i is correct and p_j is faulty. So, another possible value to reset $timer_i$ is $\max(\{susp\_level_i[j]\}_{1\leq j\leq n})$.

  The reason to reset $timer_i$ that way (instead of using s_rn_i or r_rn_i) will become clear in the last version of the algorithm (Section 6) where we will show that all the $susp\_level_i[j]$ variables can be bounded, and so all the timeout values will

[3] Let X be a non-empty set of pairs (integer, process id). The function $\min(X)$ returns the smallest pair in X, according to lexicographical order. This means that $(sl1, i)$ is smaller than $(sl2, j)$ iff $sl1 < sl2$, or $(sl1 = sl2) \wedge (i < j)$.

[4] Let us remark that an ALIVE$(rn, *)$ message that arrives after the timer has expired, but belongs to the first $(n-t)$ ALIVE$(rn, *)$ messages received by p_i, is considered by the algorithm as if it was received before the timer expiration. So, such a message cannot give rise to an erroneous suspicion.

init: **for_each** $rn \geq 1$ **do** $rec\_from_i[rn] \leftarrow \{i\}$ **end_do**;
for_each $rn \geq 1, 1 \leq j \leq n$ **do** $suspicions_i[rn, j] \leftarrow 0$ **end_do**;
$s\_rn_i \leftarrow 0$; $r\_rn_i \leftarrow 1$; $susp\_level_i \leftarrow [0, \ldots, 0]$; **set** $timer_i$ **to** 0;

task $T1$:
(1) **repeat regularly**:
% Two consecutive repeats are separated by at most β time units %
(2) $s\_rn_i \leftarrow s\_rn_i + 1$;
(3) **for_each** $j \neq i$ **do** *send* ALIVE$(s_rn_i, susp_level_i)$ *to* p_j **end_do**

task $T2$:
(4) **upon reception** ALIVE(rn, sl) **from** p_j:
(5) **for_each** k **do** $susp\_level_i[k] \leftarrow \max(susp\_level_i[k], sl[k])$ **end_do**;
(6) **if** $rn \geq r\_rn_i$ **then** $rec\_from_i[rn] \leftarrow rec\_from_i[rn] \cup \{j\}$
(7) **end_if**

(8) **when** $(timer_i$ has expired$) \wedge (|rec_from_i[r_rn_i]| \geq n - t)$:
(9) **let** $suspects = \Pi \setminus rec\_from_i[r\_rn_i]$;
(10) **for_each** j **do** *send* SUSPICION$(r_rn_i, suspects)$ *to* p_j **end_do**;
(11) **set** $timer_i$ **to** $\max(\{susp\_level_i[j]\}_{1 \leq j \leq n})$;
(12) $r\_rn_i \leftarrow r\_rn_i + 1$

(13) **upon reception** SUSPICION$(rn, suspects)$ **from** p_j:
(14) **for_each** $k \in suspects$ **do**
(15) $suspicions_i[rn, k] \leftarrow suspicions_i[rn, k] + 1$;
(16) **if** $(suspicions_i[rn, k] = n - t)$
(17) **then** $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$ **end_if**
(18) **end_do**

(19) **when** leader() **is invoked by the upper layer**:
(20) **let** ℓ **such that** $(susp\_level_i[\ell], \ell) = \min(\{(susp\_level_i[j], j)\}_{1 \leq j \leq n})$;
(21) **return** (ℓ)

Fig. 1. Algorithm for process p_i in $AS_{n,t}[\mathcal{A}^+]$

also be bounded (while the round numbers cannot be bounded). Let us notice that bounded timeout values can allow reducing stabilization time.

- Lines 13-18. When it receives a SUSPICION$(rn, suspects)$ message, p_i increases $suspicions_i[rn, k]$ for each process p_k such that $k \in suspects$ (Line 15). Moreover, if p_k is suspected by "enough" processes (here, $n - t$) during the receiving phase rn, p_i increases $susp\_level_i[k]$ (Lines 16-17)[5].

[5] It is worth noticing that the system parameter t is never explicitly used by the algorithm. This means that $(n - t)$ could be replaced by a parameter α. For the algorithm to work, α has to be a lower bound on the number of the correct processes.

4.2 Proof of the Algorithm

Lemma 1. *Let p_i be a correct process and p_j a faulty process. $susp\_level_i[j]$ increases forever.*

Lemma 2. *Let p_ℓ be a correct process that is the center of an eventual rotating t-star (i.e., it makes true $\mathcal{A}^+$). There is a time after which, for any process p_i, $susp\_level_i[\ell]$ is never increased at Line 17.*

Theorem 1. *The algorithm described in Figure 1 implements Ω in $AS_{n,t}[\mathcal{A}^+]$.*

5 An $\mathcal{A}$-Based Leader Algorithm

5.1 From $\mathcal{A}^+$ to $\mathcal{A}$

The difference between $\mathcal{A}^+$ and $\mathcal{A}$ lies on the fact that the properties A1 and A2 that define an eventual rotating t-star, have no longer to be satisfied by each round number starting from some unknown but finite number RN_0, but only by the round numbers of an infinite sequence $S = s_1, s_2, \ldots, s_k, s_{k+1}, \ldots$, that (1) starts at RN_0 (i.e., $s_1 = RN_0$), and (2) is such that $\forall k,\ s_{k+1} - s_k \leq D$, where D is a (possibly unknown) constant.

This means that, when compared to an $\mathcal{A}^+$-based algorithm, an Ω $\mathcal{A}$-based algorithm has to filter the round numbers in order to skip the irrelevant ones, i.e., the round numbers that do not belong to S. In a very interesting way, this can be attained by adding a single line (more precisely, an additional test) to the $\mathcal{A}^+$-based algorithm described in Figure 1. The corresponding $\mathcal{A}$-based algorithm is described in Figure 2 where the new line is prefixed by "$*$".

The variable $susp\_level_i[k]$ must no longer be systematically increased when there is a round number rn such that $suspicion_i[rn, k] = n - t$. This is in order to prevent such increases when rn is a round number that does not belong to the sequence S. But,

————————- The Lines 1-12 are the same as in Figure 1 ————————

(13) **upon reception** SUSPICION$(rn, suspects)$ **from** p_j:
(14) **for_each** $k \in suspects$ **do**
(15) $suspicions_i[rn, k] \leftarrow suspicions_i[rn, k] + 1$;
(16) **if** $(suspicions_i[rn, k] = n - t)$
$*$ $\wedge\ (\forall x : rn - susp\_level_i[k] < x < rn : suspicions_i[x, k] \geq n - t)$
(17) **then** $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$ **end_if**
(18) **end_do**

————————- The Lines 19-21 are the same as in Figure 1 ————————

Fig. 2. Algorithm for process p_i in $AS_{n,t}[\mathcal{A}]$

on the other side, $susp\_level_i[k]$ has to be forever increased if p_k has crashed. To attain these "conflicting" goals, the variables $susp\_level_i[k]$ and $suspicion_i[rn,k]$ are simultaneously used as follows: $susp\_level_i[k]$ is increased if $suspicion_i[rn,k] = n - t$ and, $\forall x$ such that $rn - susp\_level_i[k] < x < rn$, we have $suspicion_i[x,k] \geq n - t$. When it is satisfied, this additional condition means that p_k has been continuously suspected during "enough" rounds in order $susp\_level_i[k]$ to be increased. The exact meaning of "enough" is dynamically defined as being the round number window $[rn - susp\_level_i[k] + 1, rn]$, thereby allowing not to explicitly use the bound D (that constraints the sequence S) in the text of the algorithm.

5.2 Proof of the Algorithm

The statements of the lemmas and theorem that follow are the same as in Section 4. As $\mathcal{A}$ is weaker than $\mathcal{A}^+$ their proofs are different.

Lemma 3. *Let p_i be a correct process and p_j a faulty process. $susp\_level_i[j]$ increases forever.*

Lemma 4. *Let p_ℓ be a correct process that is the center of an eventual rotating t-star (i.e., it makes true $\mathcal{A}$). There is a time after which, for any process p_i, $susp\_level_i[\ell]$ is never increased at Line 17.*

Theorem 2. *The algorithm described in Figure 2 implements Ω in $AS_{n,t}[\mathcal{A}]$.*

6 A Bounded Variable $\mathcal{A}$-Based Leader Algorithm

When we examine the $\mathcal{A}$-based leader algorithm described in Figure 2, it appears that, for each process p_i, the size of its variables is bounded, except for variables s_rn_i, r_rn_i, and $susp\_level_i[j]$ in some cases (e.g., when p_j crashes). Since the current value of $\max(\{susp\_level_i[j]\}_{1 \leq j \leq n})$ is used by p_i to reset its timer, it follows that all the timeout values are potentially unbounded (e.g., this occurs as soon as one process crashes).

We show here that each local variable $susp\_level_i[j]$ can be bounded whatever the behavior of p_j and the time taken by the messages sent by p_j to p_i. Consequently, all the variables (except the round numbers) are bounded, be the execution finite or infinite. It follows that all the timeout values are bounded, whatever the fact that processes crash or not, and the links are timely or not. This is a noteworthy property of the algorithm. (Of course, it remains possible to use s_rn_i or r_rn_i if, due to specific application requirements, one needs to have increasing timeouts.)

6.1 Bounding all the Variables $susp\_level_i[k]$

Let us observe that if $susp\_level_i[k]$ is not the smallest value of the array $susp\_level_i$, p_i does not currently considers p_k as the leader. This means that it is not necessary to increase $susp\_level_i[k]$ when $susp\_level_i[k] \neq \min(\{susp\_level_i[j]\}_{1 \leq j \leq n})$. The proof shows that this intuition is correct.

```
(13) upon reception SUSPICION(rn, suspects) from p_j:
(14)    for_each k ∈ suspects do
(15)       suspicions_i[rn, k] ← suspicions_i[rn, k] + 1;
(16)       if   (suspicions_i[rn, k] = n − t)
  *           ∧ (∀x : rn − susp_level_i[k] < x < rn : suspicions_i[x, k] ≥ n − t)
 **           ∧ (susp_level_i[k] = min({susp_level_i[j]}_{1≤j≤n}))
(17)                   then susp_level_i[k] ← susp_level_i[k] + 1 end_if
(18)    end_do
```

Fig. 3. Algorithm with bounded variables for process p_i in $AS_{n,t}[\mathcal{A}]$

Let B be the final smallest value in the array $susp\_level_i[1..n]$, once the eventual leader has been elected. The previous observation allows us to conclude that no value in this array will ever be greater than $B + 1$, and consequently, all the values are bounded.

As for the previous algorithm (Figure 2), The resulting algorithm can be attained by adding a single line (more precisely, an additional test) to the $\mathcal{A}^+$-based algorithm described in Figure 1. This new test is described in Figure 3 where it appears at the line marked "**".

6.2 Proof and Properties of the Algorithm

Lemma 5. *Let p_ℓ be a correct process that makes true the assumption $\mathcal{A}$. There is a time after which, for any process p_i, $susp\_level_i[\ell]$ is never increased at Line 17.*

Definition 3. *Let B_j be the greatest value (or $+\infty$ if there is no such finite value) ever taken by a variable $susp\_level_i[j]$, $\forall i \in [1..n]$. Let $B = \min(B_1, \ldots, B_n)$ or $+\infty$ if all B_j are equal to $+\infty$.*

Lemma 6. *Let p_i be a correct process and p_j a faulty process. We eventually have $susp\_level_i[j] > B$.*

Theorem 3. *The algorithm described in Figure 3 implements Ω in $AS_{n,t}[\mathcal{A}]$.*

Lemma 7. $\forall\ p_i$, $\max(\{susp\_level_i[x]\}_{1\leq x\leq n}) - \min(\{susp\_level_i[x]\}_{1\leq x\leq n}) \leq 1$ *is always satisfied.*

Theorem 4. *No variable $susp\_level_i[j]$ is ever larger than $B + 1$.*

7 Conclusion

Combining the result of [3,4] with this paper we obtain the following theorem:

Theorem 5. *Consensus can be solved in any message-passing asynchronous system that has (1) a majority of correct processes ($t < n/2$), and (2) an intermittent rotating t-star.*

References

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On Implementing Omega with Weak Reliability and Synchrony Assumptions. In: 22th ACM Symposium on Principles of Distributed Computing (PODC 2003), pp. 306–314 (2003)
2. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Communication Efficient Leader Election and Consensus with Limited Link Synchrony. In: 23th ACM Symp. on Principles of Distributed Computing (PODC 2004), pp. 328–337 (2004)
3. Chandra, T.D., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM 43(2), 225–267 (1996)
4. Chandra, T.D., Hadzilacos, V., Toueg, S.: The Weakest Failure Detector for Solving Consensus. Journal of the ACM 43(4), 685–722 (1996)
5. Fernández, A., Jiménez, E., Raynal, M.: Eventual Leader Election with Weak Assumptions on Initial Knowledge, Communication Reliability, and Synchrony. In: Proc. Int'l IEEE conference on Dependable Systems and Networks (DSN 2006), pp. 166–175 (2006)
6. Fernández, A., Raynal, M.: From an Intermittent Rotating Star to a Leader. Tech Report 1810, IRISA, Rennes (2006)
7. Fischer, M.J., Lynch, N., Paterson, M.S.: Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM 32(2), 374–382 (1985)
8. Guerraoui, R.: Indulgent Algorithms.In: 19th ACM Symposium on Principles of Distributed Computing (PODC 2000), pp. 289–298 (2000)
9. Guerraoui, R., Raynal, M.: The Information Structure of Indulgent Consensus. IEEE Transactions on Computers 53(4), 453–466 (2004)
10. Hélary, J.-M., Mostéfaoui, A., Raynal, M.: Interval Consistency of Asynchronous Distributed Computations. Journal of Computer and System Sciences 64(2), 329–349 (2002)
11. Hutle, M., Malkhi, D., Schmid, U., Zhou, L.: Chasing the weakest system model for implementing Ω and consensus. In: Datta, A.K., Gradinariu, M. (eds.) SSS 2006. LNCS, vol. 4280, pp. 576–577. Springer, Heidelberg (2006)
12. Jiménez, E., Arévalo, S., Fernández, A.: Implementing unreliable failure detectors with unknown membership. Information Processing Letters 100(2), 60–63 (2006)
13. Lamport, L.: The Part-Time Parliament. ACM Trans. on Comp. Systems 16(2), 133–169 (1998)
14. Lamport, L., Shostak, R., Pease, L.: The Byzantine General Problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (1982)
15. Larrea, M., Fernández, A., Arévalo, S.: Optimal Implementation of the Weakest Failure Detector for Solving Consensus.In: Proc. 19th IEEE Int'l Symposium on Reliable Distributed Systems (SRDS 2000), pp. 52–60 (2000)
16. Malkhi, D., Oprea, F., Zhou, L.: Ω Meets Paxos: Leader Election and Stability without Eventual Timely Links. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 199–213. Springer, Heidelberg (2005)
17. Mostéfaoui, A., Mourgaya, E., Raynal, M.: Asynchronous Implementation of Failure Detectors.In: Proc. Int'l IEEE Conf. on Dependable Systems and Networks, pp. 351–360 (2003)
18. Mostéfaoui, A., Raynal, M.: Leader-Based Consensus. Parallel Processing Letters 11(1), 95–107 (2000)
19. Mostéfaoui, A., Raynal, M., Travers, C.: Crash-resilient Time-free Eventual Leadership.In: Proc. 23th Int'l IEEE Symposium on Reliable Distributed Systems, pp. 208–217 (2004)
20. Mostéfaoui, A., Raynal, M., Travers, C.: Time-free and timer-based assumptions can be combined to get eventual leadership. IEEE Transactions on Parallel and Distributed Systems 17(7), 656–666 (2006)
21. Raynal, M.: A short Introduction to Failure Detectors for Asynchronous Distributed Systems. ACM SIGACT News, Distributed Computing Column 36(1), 53–70 (2005)