From an asynchronous intermittent rotating star to an eventual leader

Antonio Fernández Anta and Michel Raynal

Abstract—Considering an asynchronous system made up of n processes and where up to t of them can crash, finding the weakest assumption 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 that direction. It has two contributions. Considering a simple and general asynchronous system model where processes generate asynchronous pulses during which they send and receive messages, it first introduces an additional assumption that allows to elect an eventual leader in all the runs that satisfy that assumption. That assumption is captured by the notion of *asynchronous intermittent rotating t-star*. An *x-star* is made up of one process p (the center of the star) plus a sequence of sets of x processes (the successive points of the star), that satisfies some properties. Intuitively, the *intermittent rotating t-star* assumption means that there are a process p, a subset of pulse numbers pn, and associated sets of processes $Q(pn)$ such that each process of $Q(pn)$ receives from p a message sent in pulse pn in a timely manner or among the first $(n - t)$ messages tagged pn it ever receives. The t-star is called *rotating* because the set $Q(pn)$ is allowed to change with pn; it is *intermittent* because it can disappear during finite periods; it is *asynchronous* because the points of a star are not required to be simultaneously at the same pulse. (That assumption combines and 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.) The second contribution of the paper is an algorithm that eventually elects a common leader in the systems that satisfy the *asynchronous 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 pulse numbers increase without bound. This means that, even in infinite executions, 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.

Index Terms—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) [4], [30]. According to the 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 each process with a *leader* variable, that contains a process id, and such that, after some finite but unknown time, the variables of all correct processes (the processes that do not fail) permanently contain the same id, that is the identity of a correct process. 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 is that, despite its very weak definition, it is powerful enough to allow solutions to fundamental problems such as the consensus problem [5]. More precisely, it has been shown to be the weakest class of failure detectors that allows consensus to be solved

• *Antonio Fernández Anta is with LADyR, GSyC, Universidad Rey Juan Carlos, 28933 Móstoles, Spain.*

• *Michel Raynal is with IRISA, Université de Rennes, Campus de Beaulieu 35 042 Rennes, France.*

The work of A. Fernández Anta was partially supported by the Spanish MICINN under grant TIN2008-06735-C02-01 and the Spanish MEC under grant PR2008-0015. The work of Michel Raynal was partially supported by the European Network of Excellence ReSIST and the French ANR project SHAMAN. Both were also supported by the Comunidad de Madrid under grant S-0505/TIC/0285.

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 [21], it cannot be solved in asynchronous distributed systems prone to even a single process crash [12]). 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 [15], [20], [26]. The second noteworthy feature of Ω lies on the fact that it allows the design of *indulgent* protocols [14]. 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 [14], [15]. 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 [12]. Direct proofs of the impossibility to implement Ω in pure crash-prone asynchronous systems can be found in [2], [27].) But thanks to indulgence, this is not totally bad news. More precisely, as Ω makes 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., when 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, i.e. when 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 message-passing 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., [4], [22]) 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 through the link 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*. (After the receiver has crashed, it is considered that the link from a correct process to a crashed process is always timely.)

Another time-based assumption has been proposed in [24], 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

1. The algorithm described in [4] uses that assumption to build an eventually perfect failure detector. The Ω protocol presented in [22] requires only the output links of the correct process with the smallest identity to be eventually timely.

$Q(\tau)$ 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 way these two notions have been combined in [17], where the notion of *eventual t-moving source* is defined. 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 [9], [18]. They consider weak assumptions on both the initial knowledge of processes and the network behavior. Protocols building Ω are presented [9], [18] 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, t, nor the ids of the other processes). An unreliable broadcast primitive allows the processes to communicate. As far as the network behavior is concerned, one of the protocols presented in [9] requires only that (1) each pair of correct processes be connected by fair lossy links, and (2) there is a correct process whose output links to the rest of correct processes are eventually timely. It is shown in [18] 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 (formed by eventually timely links and correct processes).

The message pattern approach A totally different approach to build Ω has been introduced in [25]. 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 [27] 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 pair of processes p and q such that, after some time τ (and until possibly q crashes), the round trip delay $q{-}p{-}q$ is never the slowest among all the round trips delays experienced by q.” A probabilistic analysis for the case $t = 1$ shows that such a behavioral property on the message exchange pattern is practically always satisfied [25].

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 [28]. 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. Here, the important point is that the message pattern assumption and the timely link assumption are combined at 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 [5]), 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 does exist. So, this paper is only a step in that direction.

The pulse model For convenience and simplicity of the presentation, this paper considers a simple base asynchronous computing model where processes can crash, that we call the *pulse* model. Each process (that does not crash) executes an infinite sequence of pulses. During a pulse a process first sends a message to all processes, and then receives and processes the messages it has received since the previous pulse. Then, it asynchronously proceeds to the next pulse. The processes communicate through a reliable network [4], [12] (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).

Observe that the pulse model can be directly simulated on top of the classical asynchronous send (or broadcast)/receive message-passing model. Moreover, it is easy to simulate the classical send (or broadcast)/receive model on top of the pulse model (which means that the pulse model is general enough to support many applications). Hence, the pulse model and the classical send/receive model are equivalent. It is important to notice that the pulse-based model is different from, and more general than, the *round*-based model [3], [23][3]. From a global point of view, we have the structure described in Figure 1, where the pulse model is considered as base model. On top of it, (1) the EVENTUAL LEADER module implements Ω in all its runs that satisfy the additional assumption described below, and (2) a PARTICULAR MODEL module that simulates the desired model needed by the upper layer application (send/receive, broadcast, round-based, etc.)[4].

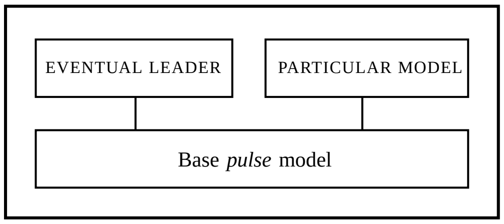

Figure 1. A particular model enriched with Ω

An eventual leader-based algorithm is *communication-efficient* if after some time, only the elected leader sends messages. Several communication-efficient leader algorithms have been designed (e.g., [2], [7]). It is important to notice that such algorithms cannot be designed in the round-based model or the pulse-based model, as in these models processes send messages in every round.

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.

3. In the round-based model, at every round r, a process receives and processes only messages sent during the very same round r.

4. The EVENTUAL LEADER module may use the even pulses of the underlying pulse model, while the PARTICULAR MODEL module uses its odd pulses.

The proposed assumption The assumption to be used to implement Ω will be denoted $\mathcal{A}$. To make the presentation easier, an assumption $\mathcal{A}^+$, of which $\mathcal{A}$ is a weakening, is first described. $\mathcal{A}^+$ is as follows. There is a correct process p and a finite pulse number α such that, at each pulse $pn \geq \alpha$, there is a set $Q(pn)$ such that $|Q(pn)| = t$, $p \notin Q(pn)$, and for each process $q \in Q(pn)$ that has not crashed, the message PULSE$(pn, -)$ 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 among the first $(n-t)$ PULSE$(pn, -)$ messages received by q. The sequence of pairs $\langle p, Q(pn)\rangle$, $pn \geq \alpha$, defines a structure that we call a t-star: its permanent center is the process p while the processes of $Q(pn)$ define its t points at pulse pn. As the set $Q()$ can change at each pulse (while p is fixed forever), we say that it is an *eventual rotating* t-star ("eventual" because there is an arbitrary finite number of pulses during which the requirement may not be satisfied). Finally, as there is no synchrony constraint on the processes that are the points of the star (the pulses of the processes are never synchronized), the star is said to be *asynchronous*.

While $\mathcal{A}^+$ defines an eventual rotating t-star that allows implementing Ω, it appears that a weakened form of that assumption, denoted $\mathcal{A}$, that defines an *intermittent* rotating t-star, is sufficient to implement an eventual leader. Basically, the difference between $\mathcal{A}^+$ and $\mathcal{A}$ is related to the notion of observation level [16]. While $\mathcal{A}^+$ considers a base level including all the pulses, $\mathcal{A}$ provides an abstraction level that allows eliminating irrelevant pulses. Of course, as it cannot be known in advance which are the relevant pulses, an $\mathcal{A}$-based algorithm has to consider a priori all the pulses and then find a way to dynamically skip the irrelevant ones.

After having introduced $\mathcal{A}^+$ and $\mathcal{A}$, the paper incrementally presents first an $\mathcal{A}^+$-based Ω algorithm, and then enriches it to obtain an $\mathcal{A}$-based Ω algorithm. The $\mathcal{A}$-based algorithm enjoys a noteworthy property, namely, all the local variables and message fields, but the pulse numbers, have a finite domain, even if the execution is infinite. This means that the timeout values used by each process eventually stabilize. From an algorithmic point of view, the proposed algorithm combines new ideas with mechanisms also used in [2], [9], [17], [25], [28]. (An early version of the proposed Ω $\mathcal{A}$-based algorithm, expressed in the classical send/receive model, can be found in [11].)

1.4 Road-map

The paper is composed of 8 sections. Section 2 presents the system model and defines the class of eventual leader oracles. Section 3 presents the additional assumptions $\mathcal{A}^+$ and $\mathcal{A}$. The presentation of the eventual leader algorithm is then done incrementally. First, Section 4 presents and proves an algorithm based on the assumption $\mathcal{A}^+$. Then, Section 5 enriches the previous algorithm to take into account the weaker assumption $\mathcal{A}$. Finally, Section 6 improves the previous algorithm to obtain an $\mathcal{A}$-based algorithm whose variables (but the pulse numbers) take a finite number of values even if the execution is infinite. Section 7 explores a framework suited to the definition of $\mathcal{A}$-like assumptions. Finally, Section 8 concludes the paper assumptions.

2 Definitions

2.1 Basic distributed system model

The underlying system model is made up of a set Π of $n \geq 2$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. The identity of p_i is its index i. The set of identities $I = \{1, \ldots, n\}$ is assumed to be known. We sometimes use p and q to denote processes.

Failure model A process can fail by crashing (halting prematurely). 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 ($1 \leq t < n$). The value t is known to the processes.

The only atomic operation of a process is the sending of a message to another process, which means that a message is sent entirely or not at all.

Pulse model A process p_i executes a sequence of *pulses*. This sequence of pulses captures its asynchronous progress (it can be seen as a local logical time). If the process is correct, that sequence is infinite. A pulse is made up of three phases.

- Send phase. During that phase, a process p_i sends a message to all processes, including itself. Such a pulse message, denoted PULSE(), carries the current pulse number (and possibly other data according to the algorithm executed by p_i).
- Receive phase. During that phase, a process p_i receives all the pulse messages that are in its input buffer. Those are the messages that have arrived since its last receive (in the previous pulse). Without loss of generality and for ease of exposition, a process always receives during a pulse the message it has sent during that pulse.
- Local computation phase. During that phase, the process p executes some local computation. When it has terminated, it proceeds to the next pulse.

Figure 2 presents the basic execution behavior of a process p_i in the pulse model.

Observe that the pulse model can be directly simulated on top of the classical asynchronous send/receive model. To do so, processes must simply maintain a pulse counter (as in Line 3 of Figure 2), label pulse messages appropriately, send the pulse counter in each pulse message, and perform the computation following the sequence of pulses, and phases within pulses as described in Figure 2. Hence, the pulse model is not stronger than the asynchronous message passing model, and any algorithm that executes on the pulse model can be executed on the asynchronous model. The pulse model is used in this work for convenience and ease of presentation.

It is also important to see that this pulse-based model is weaker (i.e., more general) than the classical asynchronous round-based model [3], [23], or the asynchronous round-by-round failure detector model (RRFD) [13]. In both these models, a message sent at round r is received at the very same round r or is never received: these models are communication-closed [8]. Differently, the pulse model is not communication closed: a message sent at pulse pn can be received by its destination process at any pulse. There is no relation linking the pulse at which a message is sent and the pulse at which it is received. More explicitly, there is no synchrony assumption hidden in the pulse notion.

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 (send, receive and local computation) take no time. Only message transfers and, for each process, the progress from one pulse to another consumes time. Moreover, these time durations are finite but arbitrary: the system is asynchronous.

The underlying communication channels 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 the two processes fails. The channels are asynchronous in the sense that they are not necessarily FIFO and there is no assumption about message transfer delays (except that they are finite).

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

```
init:  $pn_i \leftarrow 0$.

(1) repeat forever
(2) begin pulse
(3)     $pn_i \leftarrow pn_i + 1$; % Proceed to the next local pulse %
(4)     for each $p_j \in \Pi$ do send PULSE$(pn_i, i, -)$ to $p_j$ end for; % Send phase %
(5)     for each PULSE$(pn, j, -)$ received since the previous pulse do % Receive phase %
(6)         process the message PULSE$(pn, j, -)$
(7)     end for;
(8)     % Local  computation phase %
(9) end pulse.
```

Figure 2. Process behavior of process p_i under the pulse model

prove protocol properties. The range of clock values is the set of integers. We assume that each event in a run has a global time (an integer) associated, and that all references to time in the rest of the paper (like, for instance, the δ-timeliness of a message) are relative to global clock time.

Each process has a timer that can be used to accurately measure global clock intervals (possibly using the process' local clock)[5]. A process uses its timer by setting it to some value, and waiting for the timer to expire. If a process sets its timer to some value x at some global time τ, the timer will locally notify the process that it has expired at a global time $\tau + x$.

Notation In the following, $AS_{n,t}[\emptyset]$ denotes the asynchronous pulse-based model 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 pulse-based model made up of n processes among which up to $t < n$ can crash, and satisfying the additional assumption P (so, $P = \emptyset$ means no additional assumption). For convenience, sometimes $AS_{n,t}[P]$ will also be used to denote a system that runs under the corresponding computation model. Whether $AS_{n,t}[P]$ refers to a system or a model will be clear from the context.

2.2 The oracle class Ω

The oracle class Ω, informally presented in the introduction, has been defined in [5][6]. A leader oracle is a distributed entity that provides each process p_i with a variable $leader_i$ that contains a process id. A process is a leader when its id is in some variable $leader_i$ of a process p_i that has not crashed. A unique correct process is eventually elected, but there is no knowledge of when the leader is elected. Different leader variables can contain different process ids (we say "several leaders 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 [5]:

- Eventual Leadership: There is a time τ and a correct process p such that, after τ the leader variable of every correct process p_i contains p, i.e., $leader_i = p$.

Ω-based consensus algorithms are described in [15], [20], [26] 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., [4], [20]).

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], [27] ("direct proofs" means that these proofs do not rely on the impossibility of solving another given problem - such as the consensus problem [12]- in an asynchronous system).

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 oracles of the class Ω can be built. (Said differently, this means that there is an algorithm that implements Ω in all the runs of $AS_{n,t}[\emptyset]$ that satisfy $\mathcal{A}$.)

3.1 Preliminary definitions

As indicated in Section 2.1, each pulse message carries its sending pulse number. A message PULSE$(pn, -)$ 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 and pulses, not with links.) Let δ denote a bounded value

5. We assume accurate timers for simplicity. In reality for the correctness of the algorithms it is enough that the timers are asymptotically well-behaved [10].

6. The terminology used in [5] is slightly different from the one used here: While [5] uses the words "failure detector" we use the word "oracle".

(not necessarily known by the processes).

Definition 1: A message $\text{PULSE}(pn, -)$ 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 $\text{PULSE}(pn, -)$ is *winning* if it belongs to the first $(n - t)$ $\text{PULSE}(pn, -)$ messages received by its destination process.

Remark Let p be a correct process and q a faulty process. As in [1], we define that, after q has crashed, any $\text{PULSE}(pn, -)$ message sent by p to q is *δ-timely* (this is because one can always consider that the message has been received by δ time units after its sending).

Definition 3: A process p is *(α, β)-timely* if for all pulse numbers $pn \geq \alpha$, pulse messages $\text{PULSE}(pn, -)$ and $\text{PULSE}(pn+1, -)$ are sent by p separated by at most β time units.

Definition 4: Given a process p, a pulse pn of p is a *δ-flare of p* if there is a set of processes $Q(pn)$ such that

- $p \notin Q(pn)$ and $|Q(pn)| = t$,
- $\forall\, q \in Q(pn)$: the message $\text{PULSE}(pn, -)$ sent by p to q is *δ-timely* or *winning.*

3.2 The system model $AS_{n,t}[\mathcal{A}^+]$

It is important to see that, in the definitions that follow, the process p, and the bounded values α, β and δ are not known in advance, and may never be explicitly known by the processes.

Definition 5: The Assumption $\mathcal{A}^+$ is satisfied in a run if there is a correct process p, and three bounds α, β, and δ such that:

H1 Process p is (α, β)-timely,
H2 $\forall\, pn \geq \alpha$, pulse pn is a δ-flare of p.

Definition 6: In the runs that satisfy assumption $\mathcal{A}^+$, the sequence $\langle p, Q(\alpha)\rangle, \langle p, Q(\alpha + 1)\rangle, \ldots$ defines an eventual rotating t-star centered at p.

The $\mathcal{A}^+$ assumption is very flexible. One of its flexibility dimensions is related to the fact that the sets $Q()$ are not required to be the same set, i.e., if $pn \neq pn'$, $Q(pn)$ and $Q(pn')$ can be different. This is the *rotating* notion (first introduced in [17], [24] under the name *moving* set). A second flexibility dimension is the fact that two different processes $q1, q2 \in Q(pn)$ are allowed to satisfy different properties, one satisfying the "δ-timely" property, while the other satisfying the "winning" property. Finally, if q appears in $Q(pn) \cap Q(pn')$, the message $\text{PULSE}(pn, -)$ can satisfy the "δ-timely" property while the message $\text{PULSE}(pn', -)$ satisfy the "winning" property (or vice-versa).

Particular system models Albeit they have not been stated in the pulse model, it is interesting to notice that several assumptions encountered in the literature can be revisited as particular instances of the more general assumption $\mathcal{A}^+$. For this comparison we consider an underlying classical broadcast/receive asynchronous system in which the previously defined assumptions are satisfied, and we compare them with the $\mathcal{A}^+$ assumption on a simulated pulse model on top of such a system.

- If the set $Q(pn)$ is constrained to be the same for all pulse numbers $pn \geq \alpha$, and
  - Only the δ-timely notion is considered, $\mathcal{A}^+$ is equivalent to the eventual t-source assumption introduced in [2]. Consider process p and α as in assumption $\mathcal{A}^+$. All the (pulse) messages sent by p with number at least α are then received at most δ time after being sent, and hence p is an eventual t-source. Reversely, let p be an eventual t-source, then there is a time τ_0 after which all the pulse messages sent by p on its t eventually timely links are δ-timely. Setting α to be at least the first pulse number sent by p after τ_0, [H2] is satisfied. The system assumptions in [2] also guarantee that [H1] p is (α, β)-timely, for some α and β. Hence $\mathcal{A}^+$ holds.
  - Only the message winning notion is considered, $\mathcal{A}^+$ boils down to the message pattern assumption introduced in [25]. This follows trivially if a message $\text{PULSE}(pn, -)$ from process p is the response to the message $\text{PULSE}(pn, -)$ from process q. Moreover, it also holds that the message pattern assumption of [25] implies [H2]. Observe that a response may arrive before the corresponding query is broadcast, but that is not restricted in [25].
  - If both the δ-timely notion and the message winning notion are considered, $\mathcal{A}^+$ boils to the assumption used in [28]. This follows from the two previous cases.
- If the set $Q(pn)$ is allowed to vary according to the pulse numbers, and:
  - Only the δ-timely notion is considered, $\mathcal{A}^+$ is equivalent to the eventual t-moving source assumption introduced in [17].
  - Only the message winning notion is consid-

ered, $\mathcal{A}^+$ provides a t-moving message pattern assumption generalizing the assumption introduced in [25].

This shows that the assumption $\mathcal{A}^+$ can be seen as a very flexible generic assumption. Not only it includes several existing proposals, but, as we are about to see, it can be weakened, allowing thereby more runs of the underlying system to implement an Ω oracle.

3.3 The 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 pulse numbers. (To the best of our knowledge, none of the assumptions proposed so far has investigated such an assumption weakening.) $\mathcal{A}$ is $\mathcal{A}^+$, where property H2 is weakened as follows:

H2$'$ There is an infinite subsequence S of increasing pulse numbers $\alpha = \alpha_0, \alpha_1, \dots, \alpha_x, \alpha_{x+1}, \dots$ and a bound D (not necessarily known) such that $\forall\, x \geq 0$: $\alpha_{x+1} - \alpha_x \leq D$, and $\forall\, pn \in S$, pulse pn is a δ-flare of p.

When $D = 1$, $\mathcal{A}$ boils down to $\mathcal{A}^+$. So, $\mathcal{A}$ weakens $\mathcal{A}^+$ by adding another flexibility 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 for Ω to be built. This is why we say that $\mathcal{A}$ defines an *intermittent* rotating t-star. The limit imposed by $\mathcal{A}$ on this flexibility dimension is expressed by the bound D: the constraining subsequence S cannot be fully arbitrary.

4 An $\mathcal{A}^+$-based leader algorithm

This section presents and proves the correctness of an algorithm that builds an oracle 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

As a lot of other eventual leader algorithms (e.g., [2], [25]), the proposed algorithm strives to elect the process that is currently the least suspected to have crashed (if several processes have this property, their ids are used to break ties)[7].

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

- $leader_i$ contains the id of the process that p_i currently considers as the common leader. It is initialized to any process identity (e.g., 1 or i).
- pn_i is a local variable containing the current local pulse number. Its value is used to tag the pulse messages sent by p_i.
- $timer_i$ 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 pulses during which p_j is strongly suspected.
  A process p_j is "strongly suspected" during a pulse pn, if at least $(n-t)$ processes suspect it to have crashed while they execute pulse pn. Let us observe that, due to asynchrony, two processes p_x and p_y can suspect a process p_j during the same pulse but at different physical times.
- rpn_i is a local variable containing a (receive) pulse number that is used in connection with the array $rec\_from_i[1..]$. That array is such that $rec\_from_i[x]$ keeps the ids of the processes from which p_i has received and taken into account a $\text{PULSE}(x,-)$ message. At any time, only the entries $rec\_from_i[x]$ such that $x \geq rpn_i$ are meaningful for the algorithm.
- $suspicions_i[1.., 1..n]$ is an array such that $suspicions_i[px, j]$ counts, as far as the pulse px is concerned, how many processes weakly suspect p_j to have crashed. A process p_j is "weakly suspected" by p_i with respect to the pulse px, if p_i does not receive and process a $\text{PULSE}(px,-)$ message from p_j. (The predicate indicating if p_j is weakly suspected by p_i during the pulse rpn_i involves the timer of p_i, and the value of the integer $|rec\_from_i[rpn_i]|$.)
- $suspected_i$ is a local variable that contains either the default value $\perp$, or two fields, namely, a pulse number rpn and a set of processes weakly suspected during that pulse.

Process behavior The algorithm executed by each process p_i is described in Figure 3. A process p_i executes first its sending phase: it proceeds to the next pulse (Line 3), and sends a pulse message to each process p_j (including itself). In addition to its pulse number pn_i, a pulse message piggybacks three data elements: the id i of the sender, and the current values of its array $susp\_level_i$ and its local variable $suspected_i$ (Line 4).

Then, p_i receives and processes all the pulse messages that are in its input buffer (Lines 5-17). If the $\text{PULSE}(pn,-)$ message is "on time" (i.e., such that

7. 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 $(sl1 = sl2) \wedge (i < j)$.

```
init: for_each $x \geq 1$ do $rec\_from_i[x] \leftarrow \{i\}$ end for;
      for_each $x \geq 1$, $j \in I$ do $suspicions_i[x, j] \leftarrow 0$ end for;
      for_each $j \in I$ do $susp\_level_i[j] \leftarrow 0$; end for;
      $pn_i \leftarrow 0$; $rpn_i \leftarrow 1$; $suspected_i \leftarrow \perp$; $leader_i \leftarrow 1$; set $timer_i$ to $1$.

(1) repeat forever
(2) begin pulse
(3)    $pn_i \leftarrow pn_i + 1$; % Proceed to the next local pulse %
(4)    for each $p_j \in \Pi$ do send PULSE$(pn_i, i, susp\_level_i, suspected_i)$ to $p_j$ end for;
(5)    for each PULSE$(pn, j, sl, suspected)$ received since the previous pulse do
(6)       if $pn \geq rpn_i$ then $rec\_from_i[pn] \leftarrow rec\_from_i[pn] \cup \{j\}$ end_if;
(7)       for each $k \in I$ do $susp\_level_i[k] \leftarrow \max(susp\_level_i[k], sl[k])$ end for;
(8)       if $suspected \neq \perp$ then
(9)          let $(px, suspects) = suspected$;
(10)         for each $k \in suspects$ do
(11)            $suspicions_i[px, k] \leftarrow suspicions_i[px, k] + 1$;
(12)            if $(suspicions_i[px, k] = n - t)$
(13)               then $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$
(14)            end if
(15)         end for
(16)      end if
(17)   end for;
(18)   $leader_i \leftarrow \ell$ where $\ell$ is such that $(susp\_level_i[\ell], \ell) = \min(\{(susp\_level_i[j], j)\}_{j \in I})$
(19)   $suspected_i \leftarrow \perp$;
(20)   if ($timer_i$ has expired) $\wedge$ $(|rec\_from_i[rpn_i]| \geq n - t)$ then
(21)      let $suspects_i = I \setminus rec\_from_i[rpn_i]$;
(22)      $suspected_i \leftarrow (rpn_i, suspects_i)$;
(23)      $rpn_i \leftarrow rpn_i + 1$;
(24)      set $timer_i$ to $\max(\{susp\_level_i[j]\}_{j \in I})$
(25)   end if
(26) end pulse.
```

Figure 3. An Ω algorithm for $AS_{n,t}[\mathcal{A}^+]$ (code of process p_i)

$pn \geq rpn_i$), p_i updates $rec\_from_i[pn]$ (Line 6) to remember it has received on time that PULSE$(pn, -)$ message from p_j.

The pulse messages are used to gossip values, thereby allowing the receiving process to update its local state. So, p_i updates its array $susp\_level_i[1..n]$ containing the number of strong suspicions of each process p_k (Line 7). Then, if it is different from $\perp$, the value $suspected$ carried by the pulse message is meaningful and p_i processes it (Lines 9-15). That value is a pair $(px, suspects)$ where px is a pulse number and $suspects$ a set of processes. Its meaning is the following: as far as the pulse px is concerned, the sending process p_j weakly suspects all the processes p_k whose id are in the set $suspects$ (Line 22 executed by p_j). Consequently, p_i increases its weak suspicion counter $suspicions_i[px, k]$ (Line 11), and increases also its strong suspicion counter $susp\_level_i[k]$ if enough processes ("enough" is here $n - t$) have weakly suspected p_k during the pulse px (Lines 12-13).

After having processed the pulse messages it has received, the value of $leader_i$ is recomputed as it could have changed (Line 18). Then, p_i computes the new value of its local variable $suspected_i$ (Lines 19-25). It first checks if it weakly suspects processes with respect to the pulse rpn_i. In order to suspect processes, the timer has to have expired (this is to benefit from the "δ-timely message" side of the assumption $\mathcal{A}^+$), and all the messages that are winning with respect to the pulse rpn_i have to be arrived and processed (this is to benefit from the "winning message" side of the assumption). This is captured by the predicate of Line 20.

If this predicate is true, the processes from which a PULSE$(rpn_i, -)$ message has not been received and processed are weakly suspected (Lines 21-22), rpn_i is increased (Line 23), and the timer is reset (Line 24). Finally, p_i proceeds to the next pulse after an arbitrary but finite period.

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[8]. A way to ensure that the timeout value increases when

8. Let us remark that a PULSE$(pn, -)$ message that arrives after the timer has expired, but is a winning PULSE$(pn, -)$ message, 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.

there are such false suspicions, consists in adopting a conservative approach, namely, systematically increasing the timeout value. So, a correct statement to reset the timer (at Line 24) could be "**set** $timer_i$ **to** pn_i" as this pulse number monotonically increases.

As shown in the proof (Lemma 1), the local variable $susp\_level_i[j]$ is unbounded if p_i is correct and p_j is faulty. So, another possible value to reset $timer_i$ is $\max(\{susp\_level_i[j]\}_{j\in I})$. The reason to reset $timer_i$ that way (instead of using pn_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 (if needed) all the timeout values can also be bounded (while the pulse numbers cannot be bounded). Let us notice that bounded timeout values can allow reducing stabilization time.

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.

Proof Once p_j has crashed, it does no longer send PULSE() messages. Let pn be the highest pulse number used by p_j to send a PULSE() message (Line 4). Then, as no correct process p_k will ever receive from p_j a PULSE$(pn', j, -)$ message with $pn' > pn$, we have $j \notin rec\_from_k[pn']$ for all these pulse numbers pn'. So, at each $rpn_k = pn' > pn$, each correct process p_k updates $suspected_k$ to $(rpn_k, \{\cdots, j, \cdots\})$ (Line 22).

Since there are at least $(n - t)$ correct processes, each of them sends a PULSE$(pn', -, -, \{\ldots, j, \ldots\})$ message, at each pulse $pn' > pn$. As the underlying communication network is reliable, each correct process receives at least $(n-t)$ such pulse messages, and consequently executes the Lines 9-15, and $susp\_level_i[j]$ is increased. As this happens for all $pn' > pn$, $susp\_level_i[j]$ increases without bound. $\square_{Lemma\ 1}$

Let us now consider the following time notations, for any correct process p_j.

- $send\_time(j, pn)$ is the time instant at which p_j sends PULSE$(pn, -)$.
- $predicate\_time(j, pn)$ is the time instant at which the predicate of Line 20 becomes true at p_j for $rpn_j = pn$ (the Lines 21-24 are then atomically executed at that time).
- $\forall pn > 1 :\ \Delta(j, pn) = predicate\_time(j, pn) - predicate\_time(j, pn-1)$.
  The values of $\Delta(j, -)$ are the time durations that elapse, at process p_j, between consecutive executions of the Lines 21-24.

The two lemmas that follows (Lemma 2 and lemma 3) are two technical lemmas that are used in Lemma 4 to show that, if p_ℓ is a correct process that is the center of an eventual rotating t-star, there is a finite time after which no process p_i suspects p_ℓ.

Lemma 2: Let p_ℓ be a correct process that is (α, β)-timely (Definition 3). If some process p_i increases $susp\_level_i[\ell]$ without bound then, for any correct process p_j and any constant c, there is a pulse number, denoted $pn(j, c)$, such that $\forall m \geq pn(j, c)$: $predicate\_time(j, m) \geq send\_time(\ell, m) + c$.

Proof The proof is based on the following sequence of observations.

1) As $susp\_level_i[\ell]$ increases forever, it follows from the permanent gossiping issued by p_i (Lines 4 and 7) and link reliability, that $susp\_level_j[\ell]$ increases without bound.
2) As (a) the timeout value used to reset $timer_j$ is $\max(\{susp\_level_j[x]\}_{x\in I})$, (b) $susp\_level_j[\ell]$ increases without bound (previous item), and (c) $\Delta(j, pn)$ is no smaller than the timeout value used to reset $timer_j$ when $rpn_j = pn - 1$, it follows that there is a pulse number pn' such that, $\forall\ pn'' > pn'$, $\Delta(j, pn'') \geq \beta + 1$ (recall that β is the maximal duration that eventually elapses between two consecutive broadcasts of PULSE() messages by p_ℓ).
3) As p_ℓ is (α, β)-timely, it follows that $\forall\ pn > \alpha$: $send\_time(\ell, pn) - send\_time(\ell, pn-1) \leq \beta$. Moreover, $send\_time(\ell, pn) \leq send\_time(\ell, \alpha) + \beta(pn - \alpha)$.
4) The previous items 2 and 3 state that, from some pulse number $pn > \max(\alpha, pn')$, the difference between two consecutive times at which the predicate of Line 20 is satisfied at p_j is always greater than the difference between any two consecutive broadcasts of PULSE() messages by p_ℓ.
   It follows that, for any constant c, there is a pulse number $s > \max(\alpha, pn')$ such that, $\forall pn'' \geq s$, we have $predicate\_time(j, pn'') \geq send\_time(\ell, pn'') + c$. Such a pulse number s defines $pn(j, c)$.
   More explicitly, for instance we can fix $s = send\_time(\ell, \alpha) + \beta(pn' - \alpha) + pn' + c$ (where pn' is the value defined in item 2). We have the following:

$$\begin{array}{ll} & predicate\_time(j,s) \\ = & predicate\_time(j,pn') + \Sigma_{pn'<x\leq s}\Delta(j,x) \\ & \text{[Definition of } \Delta(j,x)] \\ \geq & predicate\_time(j,pn') + (\beta+1)(s-pn') \\ & \text{[Item 2]} \\ \geq & (\beta+1)(s-pn') \\ & [predicate\_time(j,pn') \geq 0] \\ = & send\_time(\ell,\alpha) + \beta(s-\alpha) + c \\ & \text{[Definition of } s] \\ \geq & send\_time(\ell,s) + c\text{[Item 3]}. \end{array}$$

$\Box_{Lemma\ 2}$

In the following, δ is the bound associated with the notion of δ-timely message used in the definition of $\mathcal{A}^+$. Due to Lemma 2, if p_ℓ is a correct process that is (α,β)-timely and any process p_i increases $susp\_level_i[\ell]$ without bound, there is a pulse number $pn(j,\delta)$ for each correct process p_j. If this happens, let pulse PN be defined as follows (where C stands for the set of the indexes of the correct processes).

$$PN = \begin{cases} \max(\{pn(j,\delta)\}_{j\in C}) & \text{if } \exists\ \delta\text{-timely messages,} \\ 0 & \text{if no } \delta\text{-timely message.} \end{cases}$$

Lemma 3: Let p_ℓ be a correct process that is (α,β)-timely. Let $susp\_level_i[\ell]$ increase without bound, for some process p_i (and hence PN is well defined). Let m be any pulse number such that $m > PN$ and pulse m is a δ-flare of p_ℓ. For any correct process p_j such that $p_j \in Q(m) \cup \{p_\ell\}$, we have $\ell \in rec\_from_j[m]$.

Proof Let us first observe that, due to the initialization, we have $\ell \in rec\_from_\ell[m]$. Let us now consider p_j such that $p_j \in Q(m)$. Due to the fact that pulse m is a δ-flare of p_ℓ, it follows that the message PULSE$(m,-)$ sent by p_ℓ is δ-timely or winning. It follows that, when the predicate of Line 20 of p_j becomes true for $rpn_j = m$, the PULSE$(m,-)$ from p_ℓ has been received before the timer expiration (case where the message is δ-timely, because $m \geq pn(j,\delta)$), or among the first $(n-t)$ PULSE$(m,-)$ messages received by p_j (case where the message is winning). Consequently, when the message PULSE$(m,-)$ from p_ℓ has been received, we had $m \geq rpn_j$, and accordingly ℓ has then been added to $rec\_from_j[m]$ (Line 6). $\Box_{Lemma\ 3}$

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.

Proof If p_i is faulty, the lemma is trivially satisfied. Assuming p_i is correct, the proof is by contradiction. So, let us assume that there is a correct process p_i that increases $susp\_level_i[\ell]$ infinitely often. Since p_ℓ satisfies assumption H1, it is (α,β)-timely, Lemma 2 applies, and then PN is well defined. As p_ℓ satisfies assumption H2, all pulse numbers $m > \max(PN,\alpha)$ are δ-flares of p_ℓ. Hence Lemma 3 holds for all these pulses. It follows that, for any $m > \max(PN,\alpha)$, no process p_j such that $p_j \in Q(m) \cup \{p_\ell\}$ piggybacks in any PULSE() message a variable $suspected_j = (m, suspects)$ such that $suspects$ includes ℓ. Due to the assumption $\mathcal{A}^+$, $p_\ell \notin Q(m)$, from which we conclude that $|Q(m) \cup \{p_\ell\}| \geq t+1$. Consequently, at any receiving phase $m > \max(PN,\alpha)$, no process p_x can receive $(n-t)$ values $suspected = (m, suspects)$ with $suspects$ containing ℓ. It follows that no local variable $susp\_level_x[\ell]$ can increase without bound. This contradicts the initial assumption and proves the lemma. $\Box_{Lemma\ 4}$

Theorem 1: The algorithm described in Figure 3 implements Ω in $AS_{n,t}[\mathcal{A}^+]$.

Proof Due to Lemma 1, for any correct process p_i and any faulty process p_k, $susp\_level_i[k]$ increases forever. Consequently, the bounded entries (if any) of any array $susp\_level_x$ are associated only with correct processes.

Due to the gossiping mechanism of the $susp\_level_x$ arrays (Lines 4 and 7) and link reliability, it follows that if there is a process p_j such that after some time no $susp\_level_i[j]$ local variable is increased at Line 13, then all the $susp\_level_i[j]$ local variables of correct processes stabilize to the same bounded value.

Let us now observe that, due to Lemma 4, there is at least one correct process p_ℓ such that, for any process p_i, $susp\_level_i[\ell]$ is eventually never increased. Consequently, at least the ℓ entry of each $susp\_level_i$ array is bounded.

Finally, as the process that is currently elected leader by a process p_i is the one that currently is locally the least suspected (the ids being used to break ties if several processes are the least suspected), it follows that there is a time after which all the processes p_i (that have not crashed) always select the same process p_x such that $susp\_level_i[x]$ is bounded, from which we conclude that eventually the same correct process is elected forever by the processes. $\Box_{Theorem\ 1}$

5 AN $\mathcal{A}$-BASED LEADER ALGORITHM

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

$\mathcal{A}^+$ states the eventual existence of a rotating t-star, while $\mathcal{A}$ states the eventual existence of an intermittent t-star. The property H2 has only to be satisfied on an infinite subsequence S of increasing pulse numbers

$\alpha_0, \alpha_1, \ldots, \alpha_x, \alpha_{x+1}, \ldots$ such that there is a bound D (not necessarily known) such that $\forall\, x \geq 0$: $\alpha_{x+1} - \alpha_x \leq D$.

This means that, when compared to an $\mathcal{A}^+$-based algorithm, an $\mathcal{A}$-based Ω algorithm has to filter the pulse numbers in order to skip the irrelevant ones, i.e., the pulse numbers that do not belong to S. This can be attained by adding a single line (more precisely, an additional test) to the $\mathcal{A}^+$-based algorithm described in Figure 3. The corresponding $\mathcal{A}$-based algorithm is described in Figure 4 where the only new line is prefixed by "$*$".

The variable $susp\_level_i[k]$ must no longer be systematically increased when there is a pulse number px such that $suspicion_i[px,k] = n-t$. This is in order to prevent such increases when px is a pulse number that does not belong to the sequence S. But, 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 $suspicion_i[px,k]$ are simultaneously used as follows: $susp\_level_i[k]$ is increased if $suspicion_i[px,k] = n-t$ and, $\forall y$ such that $\max(0, px - susp\_level_i[k]) < y < px$, we have $suspicion_i[y,k] \geq n-t$. When it is satisfied, this additional predicate means that p_k has been continuously suspected during "enough" pulses in order $susp\_level_i[k]$ to be increased. The exact meaning of "enough" is dynamically defined as being the pulse number window $[\max(0, px - susp\_level_i[k]) + 1, px - 1]$, thereby avoiding the explicit use of 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 similar as in Section 4. As $\mathcal{A}$ is weaker than $\mathcal{A}^+$ their proofs are different.

Lemma 5: Let p_i be a correct process and p_j a faulty process. $susp\_level_i[j]$ increases forever.

Proof The proof is by contradiction. Let us assume that $susp\_level_i[j]$ is bounded by X. Let pn_j be the highest pulse number used by p_j to send PULSE() messages. It follows from the algorithm that, for each $pn > pn\_j$, each correct process sends a pulse message with the variable $suspected = (pn, \{\ldots, j, \ldots\})$. Consequently, the following observation *O* holds: for each $pn > pn\_j$ and any correct process p_k, eventually $suspicions_k[pn,j]$ becomes at least $n-t$.

So, let us consider the time τ at which $susp\_level_i[j] = X$ and for each $pn \in [pn\_j + 1, \ldots, pn\_j + X]$ we have $suspicions_i[pn,j] \geq n-t$ (due to the assumption on X and Observation *O*, the time instant τ does exist). Let pn' be the smallest pulse number, such that at τ, $pn' > pn\_j + X$ and $suspicions_i[pn',j] < n-t$ (as transfer delays of the PULSE() messages are finite, there is such a pulse number pn'; on another side, it is possible that some predicates $suspicions_i[pn'',j] \geq n-t$ with $pn'' > pn'$ be satisfied). Due to Observation *O*, eventually $suspicions_i[pn',j]$ becomes equal to $n-t$. The condition stated at Line "*" becomes then true and $susp\_level_i[j]$ is increased, which contradicts the initial assumption and proves the lemma. $\Box_{Lemma\ 5}$

Lemma 6: Let p_ℓ be a correct process that is the center of an intermittent 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.

Proof The proof is by contradiction and follows similar lines as the proof of Lemma 4. If p_i is faulty, the lemma is trivially satisfied. So, let us assume that there is a correct process p_i that increases $susp\_level_i[\ell]$ infinitely often.

Claim C1: For any correct process p_j, there is a pulse number, denoted $pn(j)$, such that the value of $susp\_level_j[\ell]$ is greater than or equal to $D-1$ at time $send\_time(\ell, pn(j))$.

Proof of the claim. By the assumption used to show contradiction, $susp\_level_i[\ell]$ increases forever for the correct process p_i. It follows that eventually $susp\_level_i[\ell] \geq D-1$. Then, the claim follows from the gossiping mechanism of the $susp\_level_x$ arrays. *End of the proof of the claim C1.*

Since p_ℓ satisfies assumption H1 (it is (α,β)-timely), Lemma 2 applies, and then the value PN (defined in Section 4.2) exists and is well defined. Due to Claim *C1*, $pn(j)$ does exist for each correct process p_j. Let $PN' = \max(\{pn(j)\}_{j \in C})$ (where C stands for the set of correct processes). Moreover, let m be any pulse number greater than $\max(PN', PN + D - 1, \alpha_0)$ (recall that α_0 is the first pulse number of the sequence S, defined in the assumption $\mathcal{A}$). We have the following.

1) Using $m > \alpha_0$.
   Due to the definition of D and the fact that $m > \alpha_0$, there is some pulse α_r in the set $\{m - D + 1, ..., m\}$ that also belongs to the sequence S of pulse numbers defined in the assumption $\mathcal{A}$. It then follows from the definition of sequence S, that p_ℓ satisfies assumption H2$'$ for pulse number α_r.
2) Using $m > PN + D - 1$.
   Let us first observe that $m > PN + D - 1$ implies $\alpha_r > PN$ (where α_r is the pulse number defined in the previous item). Since p_ℓ satisfies assumption

```
——————- The lines from the beginning until Line 8 are the same as in Figure 3 —————-

(9)   let $(px, suspects) = suspected$;
(10)  for each $k \in suspects$ do
(11)      $suspicions_i[px, k] \leftarrow suspicions_i[px, k] + 1$;
(12)      if   $(suspicions_i[px, k] = n - t)$
(*)         $\wedge \big(\forall y : \max(0, px - susp\_level_i[k]) < y < px : \; (suspicions_i[y, k] \geq n - t)\big)$
(13)        then $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$;
(14)      end if
(15)  end for.

——————- The lines from Line 16 until the end are the same as in Figure 3 ———————
```

Figure 4. Algorithm for process p_i in $AS_{n,t}[\mathcal{A}]$

H2 and $\alpha_r \in S$, pulse α_r is a δ-flare of p_ℓ. Then, from Lemma 3 it follows that, for any correct process p_j such that $p_j \in Q(\alpha_r) \cup \{p_\ell\}$, we have $\ell \in rec\_from_j[\alpha_r]$. It follows that no process p_x can ever receive $(n - t)$ pulse messages with $suspected = (\alpha_r, suspects)$ and $\ell \in suspects$. Therefore, for any process p_x, it is not possible that the value of $suspicions_x[\alpha_r, \ell]$ ever reaches $n - t$.

3) Using $m > PN'$.
Since $m > PN'$, for any correct process p_j the value of $susp\_level_j[\ell]$ is at least $D - 1$ when p_j starts receiving pulse messages with $suspected = (m, suspects)$ containing ℓ (if any is ever received). Combining this observation with the previous item (on the existence of pulse α_r such that no $suspicions_x[\alpha_r, \ell]$ ever reaches $n - t$), it follows that the condition in Line "*" (namely, $\forall y : \max(0, m - susp\_level_i[k]) < y < m : suspicions_i[y, k] \geq n - t)$) is never satisfied when such a message arrives. Consequently $susp\_level_j[\ell]$ cannot be incremented in pulse m.

To conclude the proof, let us notice that this holds for any $m > \max(PN', PN + D - 1, \alpha_0)$, from which it follows that no local variable $susp\_level_x[\ell]$ can increase without bound. This contradicts the initial assumption and proves the lemma[9]. $\Box_{Lemma\ 6}$

Theorem 2: The algorithm described in Figure 4 implements Ω in $AS_{n,t}[\mathcal{A}]$.

Proof The proof is verbatim the proof of Theorem 1 after having replaced Lemmas 1 and 4 by their new version, namely Lemmas 5 and 6, respectively. $\Box_{Theorem\ 2}$

9. The reader can observe that this proof boils down to the proof of Lemma 4 when $D = 1$.

6 A BOUNDED VARIABLE $\mathcal{A}$-BASED LEADER ALGORITHM

When we examine the $\mathcal{A}$-based leader algorithm described in Figure 4, it appears that, for each process p_i, the size of its variables is bounded, except for the pulse numbers pn_i and rpn_i, and some local variable $susp\_level_i[j]$ (e.g., when p_j crashes). Since the current value of $\max(\{susp\_level_i[j]\}_{j \in I})$ is used by p_i to reset its timer, it follows that the timeout values are potentially unbounded (e.g., this occurs as soon as one process crashes).

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

It is important to notice that each run of $AS_{n,t}[\mathcal{A}]$ is characterized by a particular bound on the local variables $susp\_level_i[j]$. This bound is of the "same nature" as the particular (unknown) bounds α, β, δ and D associated with each run of $AS_{n,t}[\mathcal{A}]$. Although the determination of that bound seems very difficult (if it is ever feasible, as it depends on the bounds α, β, δ, D, and on the system asynchrony -including the order in which the pulse messages are received and processed-), it is important to notice that it practically always prevents "memory overflow" when the memory words used for the $susp\_level_i[j]$ variable are reasonably large (e.g., 64 bits), despite the fact that runs can be infinite.

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 $susp\_level_i[k] \neq \min(\{susp\_level_i[j]\}_{j \in I})$. The proof shows that this intuition is correct.

Let B be the final smallest value in the array $susp\_level_i$, 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 4), the resulting algorithm can be obtained by adding a single line (more precisely, an additional test) to the algorithm described in Figure 3. To provide the reader with a global view, the resulting algorithm associated with a pulse is entirely described in Figure 5. The new additional line is the line marked "**".

6.2 Proof and properties of the algorithm

This section shows first that the algorithm described in Figure 5 is correct, and then that it is bounded. To that end, it uses previous lemmas and a few new lemmas.

Lemma 7: Let p_ℓ be a correct process that is the center of a intermittent 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.

Proof The proof is the same as the proof of Lemma 6. In the first part of that proof, we have only to replace the occurrence of Line "*" by an occurrence of both the Lines "*" and "**". For the rest of the proof, it is sufficient to observe that the new test (Line "**") does not involve pulse numbers. $\Box_{Lemma\ 7}$

Definition 7: Let B_j be the greatest value (or $+\infty$ if there is no such finite value) ever taken by a variable $susp\_level_i[j]$, $\forall i \in I$. Let $B = \min(B_1, \dots, B_n)$ or $+\infty$ if all B_j are equal to $+\infty$.

Lemma 8: B is bounded.

Proof This lemma follows directly from the fact that no entry of $susp\_level_i[1..n]$ ever decreases and Lemma 7. $\Box_{Lemma\ 8}$

Lemma 9: Let p_i be a correct process and p_j a faulty process. We eventually have $susp\_level_i[j] > B$.

Proof The proof is a simple combination of arguments used in the proofs of the Lemmas 1 and 5.

- As in the proof of Lemma 1, there is a pulse number pn, such that for any $pn' > pn$, each correct process receives at least $(n-t)$ PULSE$(pn', -, -, \{\dots, j, \dots\})$ messages, from which it follows that the test of Line 12 is always satisfied from $pn+1$.
- As in the proof of Lemma 5, there is a pulse number from which the predicate of Line "*" is always satisfied.

It follows that there is a pulse number from which both the predicates of Line 12 and Line "*" are always satisfied.

Let us now consider a time after which $\min(\{susp\_level_i[x]\}_{x \in I}) = B$ (due to the gossiping mechanism this eventually happens). If then $susp\_level_i[j] > B$, the lemma follows (because $susp\_level_i[j]$ never decreases). Otherwise, $susp\_level_i[j] = B$. In that case, the test of Line "**" is satisfied, and accordingly $susp\_level_i[j]$ is increased. $\Box_{Lemma\ 9}$

Theorem 3: The algorithm described in Figure 5 implements Ω in $AS_{n,t}[\mathcal{A}]$.

Proof It follows from the gossiping mechanism and Lemma 8 that there is a time after which there is a process p_ℓ such that, for each non-crashed process p_i, we have $susp\_level_i[\ell] = B$. Moreover due to Lemma 9, all the processes p_x such that $susp\_level_i[x] = B$ are correct processes. It follows that all the processes eventually elect the same leader which is a correct process. $\Box_{Theorem\ 3}$

The next lemma is used in Theorem 4 to show that the algorithm is bounded.

Lemma 10: $\forall\ p_i$, $\max(\{susp\_level_i[x]\}_{x \in I}) - \min(\{susp\_level_i[x]\}_{x \in I}) \leq 1$ is always satisfied.

Proof [10] Let $INV(sl)$ be the predicate $\max(\{sl[x]\}_{x \in I}) - \min(\{sl[x]\}_{x \in I}) \leq 1$, where sl is a size n array of integers. We show that, for any process p_i, $INV(susp\_level_i)$ is invariant. The proof of the lemma is by induction. We first show that $INV(susp\_level_i)$ is initially true and then is left true each time $susp\_level_i$ is updated (at Line 7 or Line 13).

- $INV(susp\_level_i)$ is initially true (all the entries of $susp\_level_i[1..n]$ are initially equal to 0).
- Update of $susp\_level_i$ at Line 7.
  Let $sl1$ and $sl2$ be the two vector arrays from which the component-wise maximum is computed. Due to the induction assumption, both $INV(sl1)$ and

10. After having observed that the values taken by the $susp\_level_i$ arrays define a lattice, the proof of this theorem could be directly deduced from lattice theory results. We give here a slightly longer but "self-contained" proof.

```
(1) repeat forever
(2) begin pulse
(3)     $pn_i \leftarrow pn_i + 1$; % Proceed to the next local pulse %
(4)     for each $p_j \in \Pi$ do send PULSE$(pn_i, i, susp\_level_i, suspected_i)$ to $p_j$ end for;
(5)     for each PULSE$(pn, j, sl, suspected)$ received since the previous pulse do
(6)         if $pn \geq rpn_i$ then $rec\_from_i[pn] \leftarrow rec\_from_i[pn] \cup \{j\}$ end_if;
(7)         for each $k \in I$ do $susp\_level_i[k] \leftarrow \max(susp\_level_i[k], sl[k])$ end for;
(8)         if $suspected \neq \bot$ then
(9)             let $(px, suspects) = suspected$;
(10)            for each $k \in suspects$ do
(11)                $suspicions_i[px, k] \leftarrow suspicions_i[px, k] + 1$;
(12)                if $(suspicions_i[px, k] = n - t)$
(*)                    $\wedge \big(\forall y : \max(0, px - susp\_level_i[k]) < y < px : \ (suspicions_i[y, k] \geq n - t)\big)$
(**)                   $\wedge\ (susp\_level_i[k] = \min(\{susp\_level_i[j]\}_{j \in I}))$
(13)                   then $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$;
(14)                end if
(15)            end for
(16)        end if
(17)    end for;
(18)    $leader_i \leftarrow \ell$ where $\ell$ is such that $(susp\_level_i[\ell], \ell) = \min(\{(susp\_level_i[j], j)\}_{j \in I})$
(19)    $suspected_i \leftarrow \bot$;
(20)    if ($timer_i$ has expired) $\wedge$ ($|rec\_from_i[rpn_i]| \geq n - t$) then
(21)        let $suspects_i = I \setminus rec\_from_i[rpn_i]$;
(22)        $suspected_i \leftarrow (rpn_i, suspects_i)$;
(23)        $rpn_i \leftarrow rpn_i + 1$;
(24)        set $timer_i$ to $\max(\{susp\_level_i[j]\}_{j \in I})$
(25)    end if
(26) end pulse.
```

Figure 5. Algorithm with bounded variables in $AS_{n,t}[\mathcal{A}]$ (code for p_i)

$INV(sl2)$ are satisfied. Let a and b the smallest value of $sl1$ and $sl2$, respectively. Due to the induction assumption, this means that $sl1$ (resp., $sl2$) contains only a and possibly $a+1$ (resp., b and possibly $b+1$).

- Case $a = b$. The component-wise maximum of $sl1$ and $sl2$ trivially satisfies the predicate.
- Case $a < b$. We have then $a+1 \leq b$. The proof follows from the following facts:
  - $\max(a, b) = \max(a+1, b) = b$.
  - $\max(a, b+1) = \max(a+1, b+1) = b+1$.

- Update of $susp\_level_i$ at Line 13.
  Due to the test of Line "**", $INV(susp\_level_i)$ is trivially maintained when p_i executes Line 13.

$\square_{Lemma\ 10}$

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

Proof Let p_ℓ be a process such that $B_\ell = B$. Due to the Lemmas 7 and 9, p_ℓ is a correct process. Moreover, due to the gossiping mechanism, there is a time after which all the processes p_j that have not crashed are such that $\min(\{susp\_level_j[x]\}_{x \in I}\} = B$. The theorem then follows from Lemma 10. $\square_{Theorem\ 4}$

7 A FRAMEWORK FOR A FAMILY OF $\mathcal{A}$-LIKE ASSUMPTIONS

This section investigates a framework, denoted $\mathcal{A}_{f,g}$, from which assumptions similar to $\mathcal{A}$ can be designed. Its aim is to allow the delays experienced by timely messages or the maximal distance D between the appearance of the points of the t-stars to grow unbounded. This framework is based on two additional functions $f()$ and $g()$ that allow generalizing the definition of the sequence S, and the the notion of δ-timely message, respectively. More precisely, we have the following.

The function $f()$ This function is from the set of pulse numbers into the set of integers such that, for any pulse number pn, we have $f(pn) > -D$. (As D is not known, it is always possible to define $f()$ such that $f(pn) \geq 0$ for any pulse number pn.)

The motivation for the function $f()$ is to "weaken" the constraint $\alpha_{x+1} - \alpha_x \leq D$ used to define the infinite subsequence S of pulse numbers $\alpha_0, \alpha_1, \ldots$ that appears in the statement of the assumption $\mathcal{A}$. This constraint can be made specific to each pulse number, reformulating it as follows: $\forall x \geq 0 : \ \alpha_{x+1} - \alpha_x \leq D + f(\alpha_x)$, thereby allowing the the intermittence periods during which the t-star "disappears" to grow without bound.

It is easy to see that the particular function

$\forall pn: \ f(pn) = 0$ corresponds to the basic constraint used in the assumption $\mathcal{A}$. The reader can also check that the particular function $\forall pn: \ f(pn) = 1 - D$ does correspond to the assumption $\mathcal{A}^+$.

The funtion $g()$ This function is from the set of pulse numbers into the time domain. More precisely, $g(pn)$ defines a time duration. The idea that underlies the introduction of $g()$ is to "weaken" the δ-timely message notion (Definition 1) in order to add a dynamic dimension to the unknown bound δ. This notion is replaced by the following:

Definition 8: A message $\text{PULSE}(pn, -)$ is (δ, g)-*timely* if it is received by its destination process at most $\delta + g(pn)$ time units after it has been sent.
If $\forall \ pn: \ g(pn) = 0$, the (δ, g)-timely message notion boils down to δ-timely message.

System model $AS_{n,t}[\mathcal{A}_{f,g}]$ Assuming two functions $f()$ and $g()$ as defined above, the system model $AS_{n,t}[\mathcal{A}_{f,g}]$ is $AS_{n,t}[\mathcal{A}]$ where (1) the constraint $\alpha_{x+1} - \alpha_x \leq D$ is replaced by $\alpha_{x+1} - \alpha_x \leq D + f(\alpha_x)$ in the definition of the sequence S, and (2) for all the $\text{PULSE}()$ messages, the notion of δ-timely message is replaced by the notion of (δ, g)-timely message.

An $\mathcal{A}_{f,g}$**-based algorithm** Assuming the processes know the functions $f()$ and $g()$, a very simple modification of the $\mathcal{A}$-based algorithm described in Figure 5 provides an $\mathcal{A}_{f,g}$-based algorithm that elects an eventual leader. These modifications are the following (the new parts are underlined):

- The timer resetting (Line 24) has now to take into account the function $g()$ applied to the next pulse number. Hence, Line 24 becomes: **set** $timer_i$ **to** $\max(\{susp\_level_i[j]\}_{j \in I})\underline{+g(rpn_i + 1)}$.
- The definition of the interval used in the test of Line "*" has to take into account the new constraint $\alpha_{x+1} - \alpha_x \leq D + f(\alpha_x)$. This interval is now: $\forall y : \max(0, px - (susp\_level_i[k]\underline{+f(px)})) < y < px$.

While D and δ are unknown bounds, the functions $f()$ and $g()$ appear explicitly in the algorithm and consequently have to be known by the processes. As we have seen, the particular functions $\forall pn$, $f(pn) = 0$ and $g(pn) = 0$ give rise to $\mathcal{A}$. The proof of the $\mathcal{A}_{f,g}$-based algorithm is basically the same as the proof of the $\mathcal{A}$-based algorithm.

8 Concluding remarks

The content of the paper This paper has first proposed a simple yet general system model in which an eventual leader can be elected as soon as is verified an assumption called *asynchronous intermittent rotating t-star*. That assumption states the existence of a process p (the center of the star) and logical times (pulses) such that, for a subset of these pulses pn, there are sets $Q(pn)$ of t processes and each process of each $Q(pn)$ receives from p a message tagged pn in a timely manner or among the first $(n-t)$ messages tagged pn it ever receives. We have seen that this assumption, not only combines several assumptions already proposed, but generalizes them as it also includes new assumptions not previously stated in the literature. An appendix explores a framework from which $\mathcal{A}$-like assumptions could be generated.

The paper has also presented an algorithm based on that *asynchronous intermittent rotating t-star* assumption. The presentation has voluntarily been done in a methodological and incremental way. That algorithm enjoys several noteworthy properties. From a design point of view, it is relatively simple (and design simplicity is a first-class property). From a coverage assumption point of view [29], it provides a better coverage than any algorithm based on a single base assumption (such as the t-moving source assumption or the message pattern assumption). Finally, except for the pulse numbers, the proposed algorithm uses only bounded variables, which means that, eventually, even the timeout values stop increasing.

Last but not least, combining the result of [4], [5] with this paper we obtain the following theorem:

Theorem 5: The consensus problem can be solved in any asynchronous message-passing system $AS_{n,t}[\mathcal{A}]$ that has a majority of correct processes ($t < n/2$).

The contract framework While the assumptions $\mathcal{A}$ and $\mathcal{A}_{f,g}$ are steps into identifying the weakest system requirements to eventually elect a leader, they are not the final step. In fact, these assumptions, and most assumptions in the literature on eventual leader election, fit within a common framework, which we call a *contract framework*[11].

In this framework, each process signs a "contract" with the other processes regarding its behavior, as seen by them (e.g., the messages of the process will be timely, or they will arrive in some order, etc.). The system assumptions must allow the contract specified by at least one correct process to be respected with respect to a subset of other processes, and the job of an algorithm is then to elect a leader when this happens. This *contract framework* seems to be a promising approach to capture the system requirements for eventual leader election, that we intend to investigate in the future.

11. The distributed round-based models such as the ones described in [6], [13], [19] can be seen as implicitly referring to a *contract* notion that states, for every round, which messages have to be received by each process.

Acknowledgment

The authors want to thank the referees for their constructive comments.

References

[1] Aguilera M.K., Delporte-Gallet C., Fauconnier H. and Toueg S., On Implementing Omega in Systems with Weak Reliability and Synchrony Assumptions. *Distributed Computing*, 21(4):285-314, 2008.

[2] Aguilera M.K., Delporte-Gallet C., Fauconnier H. and Toueg S., Communication Efficient Leader Election and Consensus with Limited Link Synchrony. *23th ACM Symp. on Principles of Distributed Computing (PODC'04)*, pp. 328-337, 2004.

[3] Attiya H. and Welch J., Distributed Computing, Fundamentals, Simulation and Advanced Topics (Second edition). *Wiley Series on Parallel and Distributed Computing*, 414 pages, 2004.

[4] Chandra T.D. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[5] Chandra T.D., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685-722, 1996.

[6] Charron B. and Schiper A., Harmful Dogmas in Fault-tolerant Distributed Computing. *ACM Sigact News, Distributed Computing Column*, 38(1):53-61, 2007.

[7] Delporte-Gallet C., Devismes S. and Fauconnier H., Robust Stabilizing Leader Election. *Proc. 9th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'07)*, Spinger-Verlag #4838, pp. 219-233, 2007.

[8] Elrad T.E. and Francez N., Decomposition of Distributed Programs into Communication-Closed Layers. *Science of Computer Programming*, 2(3):155-173, 1982.

[9] Fernández A., Jiménez E. and Raynal M., Eventual Leader Election with Weak Assumptions on Initial Knowledge, Communication Reliability, and Synchrony. *Proc. Int'l IEEE conference on Dependable Systems and Networks (DSN'06)*, pp. 166-175, 2006.

[10] Fernández A., Jiménez E., Trédan G. and Raynal M., A Timing Assumption and two t-resilient Protocols for Implementing an Eventual Leader Service in Asynchronous shared Memory Systems. *Algorithmica*, in press. DOI 10.1007/s00453-008-9190-2.

[11] Fernández A. and Raynal M., From an Intermittent Rotating Star to a Leader. *Proc. 11th Int'l Conference On Principles Of Distributed Systems (OPODIS'07)*, Springer-Verlag LNCS #4878, pp. 189-203, 2007.

[12] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[13] Gafni E., Round-by-round Fault Detectors: Unifying Synchrony and Asynchrony. *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, 1998.

[14] Guerraoui R., Indulgent Algorithms. *19th ACM Symposium on Principles of Distributed Computing, (PODC'00)*, pp. 289-298, 2000.

[15] Guerraoui R. and Raynal M., The Information Structure of Indulgent Consensus. *IEEE Transactions on Computers,* 53(4):453-466, 2004.

[16] Hélary J.-M., Mostéfaoui A. and Raynal M., Interval Consistency of Asynchronous Distributed Computations. *Journal of Computer and System Sciences*, 64(2):329-349, 2002.

[17] Hutle M., Malkhi D., Schmid U. and Zhou L., Chasing the Weakest System Model for Implementing Ω and Consensus. *Brief Announcement, Proc. 8th Int'l Symposium on Stabilization, Safety and Security in Distributed Systems (SSS'06)*, LNCS #4280, pp. 576-577, 2006. Full version to appear in *IEEE Transactions on Dependable and Secure Computing*, 2009.

[18] Jiménez E., Arévalo S. and Fernández A., Implementing Unreliable Failure Detectors with Unknown Membership. *Information Processing Letters*, 100(2):60-63, 2006.

[19] Keidar I. and Shraer A., How to Choose a Timing Model. *IEEE Transactions on Parallel Distributed Systems*, 19(10):1367-1380, 2008.

[20] Lamport L., The Part-Time Parliament. *ACM Transactions on Computer Systems*, 16(2):133-169, 1998.

[21] Lamport L., Shostak R. and Pease L., The Byzantine General Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382-401, 1982.

[22] Larrea M., Fernández A. and Arévalo S., Optimal Implementation of the Weakest Failure Detector for Solving Consensus. *Proc. 19th IEEE Int'l Symposium on Reliable Distributed Systems (SRDS'00)*, pp. 52-60, 2000.

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

[24] Malkhi D., Oprea F. and Zhou L., Ω Meets Paxos: Leader Election and Stability without Eventual Timely Links. *Proc. 19th Int'l Symposium on Distributed Computing (DISC'05)*, Springer Verlag LNCS #3724, pp. 199-213, 2005.

[25] Mostéfaoui A., Mourgaya E., and Raynal M., Asynchronous Implementation of Failure Detectors. *Proc. Int'l IEEE Conf. on Dependable Systems and Networks*, pp. 351-360, 2003.

[26] Mostéfaoui A. and Raynal M., Leader-Based Consensus. *Parallel Processing Letters*, 11(1):95-107, 2000.

[27] Mostéfaoui A., Raynal M. and Travers C., Crash-resilient Time-free Eventual Leadership. *Proc. 23th Int'l IEEE Symposium on Reliable Distributed Systems*, pp. 208-217, 2004.

[28] Mostéfaoui A., Raynal M. and 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.

[29] Powell D., Failure Mode Assumptions and Assumption Coverage. *Proc. of the 22nd Int'l Symposium on Fault-Tolerant Computing (FTCS-22)*, pp.386-395,Boston (MA), 1992.

[30] Raynal M., A Short Introduction to Failure Detectors for Asynchronous Distributed Systems. *ACM SIGACT News, Distributed Computing Column*, 36(1):53-70, 2005.

Antonio Fernández Anta is a Professor at the Universidad Rey Juan Carlos in Madrid, where he has been on the faculty since 1998. Previously, he was on the faculty of the Universidad Politecnica de Madrid. He graduated in Computer Science from the Universidad Politecnica de Madrid in 1991. He got a PhD in Computer Science from the University of Southwestern Louisiana in 1994 and was a postdoc at the Massachusetts Institute of Technology from 1995 to 1997. He is currently Senior Member of IEEE and ACM.

Michel Raynal received the doctorat d'Etat in computer science from the University of Rennes, France, in 1981. He is currently a professor of computer science in IRISA (CNRSINRIA- University joint computing research laboratory located in Rennes), where he founded a research group on distributed algorithms in 1983. His research interests include distributed algorithms, distributed computing systems and dependability. His main interest lies in the fundamental principles that underlie the design and the construction of distributed computing systems. He has been Principal Investigator of a number of research grants in these areas, and has been invited by many universities all over the world to give lectures and tutorials on distributed algorithms and fault-tolerant distributed computing systems. He belongs to the editorial board of several international journals. He has published more than 115 papers in journals and more than 230 papers in international conferences. He has also written seven books devoted to parallelism, distributed algorithms and systems (MIT Press and Wiley). He has served in program committees for more than 100 international conferences (including PODC, DISC, ICDCS, DSN, SRDS, etc.) and chaired the program committee of more than 15 international conferences. In 2002-2004, he chaired the steering committee leading the DISC symposium series. Michel Raynal got the IEEE ICDCS best paper Award three times in a row: 1999, 2000 and 2001. He also got the SSS 2009 best paper award. Recently, he co-chaired the SIROCCO 2005 conference, and the IWDC 2005 conference. Michel Raynal was general co-chair of ICDCS 2006. He is the program co-chair of OPODIS 2009 and conference co-chair of ICDCN 2010.