From an intermittent rotating star to a leader

Antonio Fernández, Michel Raynal

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

PUBLICATION
INTERNE
N° 1810

FROM AN INTERMITTENT ROTATING STAR TO A LEADER

ANTONIO FERNÁNDEZ MICHEL RAYNAL

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

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

From an intermittent rotating star to a leader

Antonio Fernández* Michel Raynal**

Systèmes communicants

Publication interne n ° 1810 — Août 2006 — 19 pages

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 being 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 several noteworthy properties. From a design point of view, it is simple. 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.

Key-words: 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.

(Résumé : tsvp)

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

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

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

D’une étoile qui bouge et clignote à l’élection d’un leader

Résumé : Ce rapport présente un protocole d’élection d’un leader inéluctable dans un système réparti défini par des hypothèses de synchronisme très faible.

Mots clés : Systèmes répartis asynchrones, Tolérance aux fautes, Crash de processus, Oracle oméga, Détection de fautes, Leader inéluctable, Synchronie partielle.

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 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 [13], it cannot be solved in asynchronous distributed systems prone to even a single process crash [6]). Basically, an Ω-based consensus algorithm uses the eventual leader to impose a value to all the processes, thereby providing the algorithm liveness. As the algorithm does not know when the eventual leader is elected, its main work is guaranteeing that no two different values can be decided before the eventual leader is elected. Leader-based consensus protocols can be found in [8, 12, 17].

The second noteworthy feature of Ω lies in the fact that it allows the design of *indulgent* protocols [7]. 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 [7, 8].

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 [6]. Direct proofs of the impossibility to implement Ω in pure crash-prone asynchronous systems can be found in [2, 18].) 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 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., [14]) 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 an 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 [15] 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 way these two notions have been combined in [10] 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, 11]. They consider weak assumptions on both the initial knowledge of processes and the network behavior. Protocols building Ω are presented [5, 11] that assume the initial knowledge of each process is limited to its identity and the fact that no two identities are the same (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 [11] 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 [16]. 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 [18] 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 [16].

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 [19]. This combination shows that Ω can be implemented as soon as there is a correct process p such that there is 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 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,

[1] Actually, the Ω protocol presented in [14] only requires that the output links of the correct process with the smallest identity to be eventually timely.

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, 6]. (Fair lossy links could be used instead of reliable links, but we do not consider that possibility in order to keep the presentation simple.) [2] 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 the the message ALIVE(rn) sent by p is received by q (2) at most δ time units after it has been sent (the corresponding bound δ can be unknown), or (3) among the first $(n-t)$ ALIVE(rn) messages received by q (i.e., it is 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 rotating t-star* ("eventual" because there is an arbitrary finite number of round numbers during which the requirement can be not 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 [9]. 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 an $\mathcal{A}$-based algorithm that builds a failure detector oracle of the class Ω. That algorithm enjoys a noteworthy property, namely, in an infinite execution, only the round numbers

[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.

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, 10, 16, 19].

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 round numbers) take a finite number of values be the execution finite or infinite. Section 7 presents an extension of the system model. Finally, Section 8 concludes the paper.

2 Definitions

2.1 Basic distributed system model

We consider a system formed by a finite set Π of $n \geq 2$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. 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). It 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 $p \to q$ and $q \to p$. 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 p.

Ω-based consensus algorithms are described in [8, 12, 17] 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, 12]).

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, 18] ("direct proofs" means that they are not based on the impossibility of asynchronously solving a given problem such as the consensus problem [6]).

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 $|Q(rn)| \geq t$ ($\{p\} \cup Q(rn)$ 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 [10, 15] under the name *moving* set). A second dynamicity notion is the fact that two points of the star $\{p\} \cup Q(rn)$ (e.g., $p \to q1$ and $p \to q2$), 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 these other messages, even if they use the same links as the ALIVE messages.

Particular system models It is interesting to notice that $\mathcal{A}^+$ includes assumptions previously proposed.

- If the set $Q(rn)$ is constrained to be the same for all round numbers $rn \geq RN_0$, and
  - Only the properties (1) or (2) are satisfied, $\mathcal{A}^+$ boils down to the eventual t-source assumption introduced in [2].
  - Only the properties (1) or (3) are satisfied, $\mathcal{A}^+$ boils down to the message pattern assumption introduced in [16].
  - The properties (1), (2) or (3) are satisfied, $\mathcal{A}^+$ boils to the assumption used in [19].
- If the set $Q(rn)$ can vary according to the round numbers, and:
  - Only the properties (1) or (2) are satisfied, $\mathcal{A}^+$ boils down to the eventual t-moving source assumption introduced in [10].
  - Only the properties (1) or (3) are satisfied, $\mathcal{A}^+$ provides a t-moving message pattern assumption generalizing the assumption introduced in [16].

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 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, 16]-): 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 suspects p_j to have 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 15) 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 also be bounded (while the round numbers cannot be bounded). Let us notice that bounded timeout values can allow reducing stabilization time.

[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.

- 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]

```
init: for_each rn ≥ 1 do rec_from_i[rn] ← {i} end_do;
      for_each rn ≥ 1, 1 ≤ j ≤ n do suspicions_i[rn, j] ← 0 end_do;
      s_rn_i ← 0; r_rn_i ← 1; susp_level_i ← [0, ..., 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 ← s_rn_i + 1;
(3)   for_each j ≠ 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] ← max(susp_level_i[k], sl[k]) end_do;
(6)   if rn ≥ r_rn_i then rec_from_i[rn] ← rec_from_i[rn] ∪ {j}
(7)   end_if

(8) when (timer_i has expired) ∧ (|rec_from_i[r_rn_i]| ≥ n − t):
(9)   let suspects = Π \ 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≤j≤n});
(12)  r_rn_i ← r_rn_i + 1

(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)
(17)                then susp_level_i[k] ← 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[ℓ], ℓ) = min({(susp_level_i[j], j)}_{1≤j≤n});
(21)  return (ℓ)
```

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

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 not send new ALIVE() messages. Let rn be the highest round number used by p_j to send an ALIVE() message (Line 3). Then, as no correct process p_k will ever receive from p_j an ALIVE$(rn', *)$ message with $rn' > rn$, we have $j \notin rec\_from_k[rn']$ for all these round numbers rn'. So, for each $r\_rn_k = rn' > rn$, each correct process p_k sends SUSPICION$(rn', \{\ldots, j, \ldots\})$ to each process (Line10).

Since there are at least $(n - t)$ correct processes, each of them sends SUSPICION$(rn', \{\ldots, j, \ldots\})$, and the links are reliable, it follows that each correct processes receives at least $(n - t)$ such messages and consequently executes Line 17. Hence, $susp\_level_i[j]$ is increased. As this happens for all $rn' > rn$, $susp\_level_i[j]$ increases without bound.

$\square_{Lemma\ 1}$

Lemma 2 *Let p_ℓ be a correct process that is the center of an eventual rotating t-star (i.e., it 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.*

[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.

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 $susp\_level_i[\ell]$ at Line 17 infinitely often. Let us consider the following time definitions:

- $send\_time(x, rn)$ is the instant time at which p_x sends ALIVE$(rn, *)$.
- $predicate\_time(y, rn)$ is the time instant at which the predicate of Line 8 becomes true at p_y for $r\_rn_y = rn$ (the Lines 9-12 are then atomically executed at that time).
- $\forall rn > 1 : \Delta(j, rn) = predicate\_time(j, rn) - predicate\_time(j, rn - 1)$.

Claim C1: For any correct process p_j and any constant c, there is a round number, denoted $rn(j, c)$, such that $predicate\_time(j, rn(j, c)) > send\_time(\ell, rn(j, c)) + c$.
Proof of the claim. 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 3 and 5) and link reliability, that $susp\_level_j[\ell]$ increases without bound.
2. As (1) the timeout value used to reset $timer_i$ is $\max(\{susp\_level_j[x]\}_{1\leq x\leq n})$, (2) $susp\_level_j[\ell]$ increases without bound (previous item), and (3) $\Delta(j, rn)$ is no smaller than the last timeout value used to reset $timer_i$, it follows that there is a round number rn' such that, $\forall rn'' > rn'$, $\Delta(j, rn'') \geq \beta + 1$ (the maximal duration that elapses between two consecutive broadcasts of ALIVE() messages by a process).
3. $\forall rn > 1$, we have $send\_time(\ell, rn) - send\_time(\ell, rn-1) \leq \beta$. Moreover, $send\_time(\ell, rn) \leq \tau_\ell + \beta(rn - 1)$, where τ_ℓ is the sending time by p_ℓ of its first ALIVE() message. This directly follows from the very definition of β.
4. The previous items 2 and 3 state that, from some round number rn', the difference between two consecutive times at which the predicate of Line 8 is satisfied at p_j is always greater than the difference between any two consecutive broadcasts of ALIVE() messages by p_ℓ.

   It follows that, for any constant c, there is a round number $s > rn'$ such that, $\forall rn'' \geq s$, we have $predicate\_time(j, rn'') > send\_time(\ell, rn'') + c$. Such a round number s defines $rn(j, c)$.

   More explicitly, for instance we can fix $s = rn'(\beta + 1) + \tau_\ell + c$ (where rn' is the value defined in item 2, and τ_ℓ is the time defined in item 3). We have the following:

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

End of the proof of the claim C1.

In the following, δ is the bound associated with the notion of δ-timely message used in the definition of $\mathcal{A}^+$. Due to the claim *C1*, the $rn(j, \delta)$ round number does exist for each correct process p_j. Let (where C stands for the set of correct processes):

$$RN_1 = \begin{cases} \max(\{rn(j, \delta)\}_{j\in C}) & \text{if there are } \delta\text{-timely messages,} \\ 0 & \text{if there is no } \delta\text{-timely message.} \end{cases}$$

Claim C2: Let m be any round number greater than $\max(RN_1, RN_0)$ (RN_0 is the round number defined in the assumption $\mathcal{A}^+$). For any correct process p_j such that $j \in Q(m) \cup \{\ell\}$, we have $\ell \in rec\_from_j[m]$.
Proof of the claim. 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 $j \in Q(m)$. Due to the assumption $\mathcal{A}^+$, and the fact that $m > RN_0$, it follows that the message ALIVE$(m, *)$ sent by p_ℓ is δ-timely or winning. It follows that, when the predicate of Line 8 of p_j becomes true

for $r\_rn_j = m$, the ALIVE$(m, *)$ from p_ℓ has been received before the timer expiration (case where the message is δ-timely because $m \geq rn(j, \delta)$), or among the first $(n - t)$ ALIVE$(m, *)$ messages received by p_j (case where the message is winning). Consequently, when the message ALIVE$(m, *)$ from p_ℓ has been received, we had $m \geq r\_rn_j$, and accordingly ℓ has then been added to $rec\_from_j[m]$ (Line 6). *End of the proof of the claim C2.*

It follows from the claim *C2* that, for any $m > \max(RN_1, RN_0)$, no process p_j such that $j \in Q(m) \cup \{\ell\}$ sends (at Line 10) a message SUSPICION$(m, suspects)$ such that $suspects$ includes ℓ. Due to the assumption $\mathcal{A}^+$, $\ell \notin Q(m)$, from which we conclude that $|Q(m) \cup \{\ell\}| \geq t + 1$. Consequently, at any receiving phase $m > \max(RN_1, RN_0)$, no process p_x can receive $(n - t)$ SUSPICION$(m, suspects)$ 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. □ $_{Lemma\ 2}$

Theorem 1 *The algorithm described in Figure 1 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 3 and 5) 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 17, then all $susp\_level_i[j]$ local variables stabilize to the same bounded value.

Let us now observe that, due to Lemma 2, there is at least one correct process p_ℓ such that, for any process p_i, $susp\_level_i[\ell]$ is never increased at Line 17. Consequently, at least the ℓ entry of each $susp\_level_i$ array is bounded.

Finally, as the process that is currently elected by a process p_i is the one that currently is locally the least suspected (the ids being used to do a tie-break if several processes are the less 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. □ $_{Theorem\ 1}$

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

(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)
(17)                  then susp_level_i[k] ← susp_level_i[k] + 1 end_if
(18)    end_do

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

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

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, 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.*

Proof The proof is by contradiction. Let us assume that $susp\_level_i[j]$ is bounded by X. Let rn_j be the highest round number used by p_j to send ALIVE() messages. It follows from the algorithm that, for each $rn > rn\_j$, each correct process sends a SUSPICION$(rn, \{\dots, j, \dots\})$ message. Consequently (Observation *O*), for each $rn > rn\_j$ and any correct process p_k, eventually $suspicions_k[rn, j]$ becomes $\geq n - t$.

So, let us consider the time τ at which $susp\_level_i[j] = X$ and for each $rn \in [rn\_j + 1, \dots, rn\_j + X]$ we have $suspicions_i[rn, j] \geq n - t$ (due to the assumption on X and Observation *O*, the time instant τ does exist). Let rn' be the smallest round number, such that at τ, $rn' > rn\_j + X$ and $suspicions_i[rn', j] < n - t$ (as transfer delays of the SUSPICION() messages are finite, there is such a round number rn'; on another side, it is possible that some predicates $suspicions_i[rn'', j] \geq n - t$ with $rn'' > rn'$ be satisfied). Due to Observation *O*, eventually $suspicions_i[rn', j]$ 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. □ $_{Lemma\ 3}$

Lemma 4 *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.*

Proof The proof is by contradiction and follows similar lines as the proof of Lemma 2. So, let us assume that there is a correct process p_i that increases $susp\_level_i[\ell]$ at Line 17 infinitely often (if p_i is faulty, the lemma is trivially satisfied). The notations $send\_time(x, rn)$, $predicate\_time(y, rn)$, and $\Delta(j, rn)$ have the same meaning as in the proof of Lemma 2.

Let us first notice that the Claim *C1* stated and proved in the proof of Lemma 2 still holds. That claim states that for any correct process p_j and any constant c, there is a round number, that we denote $rn(j, c)$, such that $predicate\_time(j, rn(j, c)) > send\_time(\ell, rn(j, c)) + c$. So, let RN_1 be the round number defined in the proof of that lemma.
Claim C3: For any correct process p_j, there is a round number, denoted $rn(j)$, such that the value of $susp\_level_j[\ell]$ is greater than or equal to $D - 1$ at time $send\_time(\ell, rn(j))$.
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 C3.*

Due to *C3*, $rn(j)$ does exist for each correct process p_j. Let $RN_2 = \max(\{rn(j)\}_{j \in C})$ (where C stands for the set of correct processes). Moreover, let m be any round number greater than $\max(RN_2, RN_1 + D - 1, RN_0)$ (recall that RN_0 is the round number defined in the assumption $\mathcal{A}$). We have the following.

1. Using $m > RN_0$.
   Due to the definition of D and the fact that $m > RN_0$, there is some round r in the set $\{m - D + 1, ..., m\}$ that also belongs to the sequence S of round numbers defined in the assumption $\mathcal{A}$. It then follows from the

definition of sequence S, that there is a set of processes $Q(r)$ such that each of them satisfies properties A1 and A2 with respect to p_ℓ.

2. Using $m > RN_1 + D - 1$.
Let us first observe that $m > RN_1 + D - 1$ implies $r > RN_1$ (where r is the round number defined in the previous item). Using then the same arguments as in the proof of the claim *C2* (stated and proved in Lemma 2), we can prove that, for any correct process p_j such that $j \in Q(r) \cup \{\ell\}$, we have $\ell \in rec\_from_j[r]$. It follows that no process p_x can ever receive $(n-t)$ SUSPICION$(r, suspects)$ messages such that $suspects$ contains ℓ. Therefore, for any process p_x, it is not possible that the value of $suspicions_x[r, \ell]$ ever reaches $n-t$.

3. Using $m > RN_2$.
Since $m > RN_2$, for any correct process p_j the value of $susp\_level_j[\ell]$ is at least $D-1$ when p_j starts receiving SUSPICION$(m, suspects)$ messages containing ℓ (if any is ever received). Combining this observation with the previous item (on the existence of round r such that no $suspicions_x[r, \ell]$ ever reaches $n-t$), it follows that the condition in Line "*" (namely, $\forall y : \ m - susp\_level_i[k] < y < \ m : suspicions_i[y, k] \geq n - t$)) is never satisfied when such a message arrives. Consequently $susp\_level_j[\ell]$ cannot be incremented in round m.

To conclude the proof, let us notice that this holds for any $m > \max(RN_2, RN_1 + D - 1, RN_0)$, from which it follows that that no local variable $susp\_level_x[\ell]$ can increase without bound. This contradicts the initial assumption and proves the lemma[6]. $\square_{Lemma\ 4}$

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

Proof The proof is verbatim the proof of Theorem 1 after having replaced Lemmas 1 and 2 by their new version, namely Lemmas 3 and 4, respectively. $\square_{Theorem\ 2}$

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 $susp\_level_i[1..n]$, 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.

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.

The resulting algorithm is described in Figure 3. It is the same as the previous with only one additional test (new line marked "**"). To ease its reading and provide a global view, the algorithm is given entirely.

[6] The reader can observe that this proof boils down to the proof of Lemma 2 when $D = 1$.

```
init: for_each rn ≥ 1 do rec_from_i[rn] ← {i} end_do;
      for_each rn ≥ 1, 1 ≤ j ≤ n do suspicions_i[rn, j] ← 0 end_do;
      s_rn_i ← 0; r_rn_i ← 1; susp_level_i ← [0, ..., 0]; set timer_i to 0;

task T1:
(1) repeat regularly:
(2)     s_rn_i ← s_rn_i + 1;
(3)     for_each j ≠ 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] ← max(susp_level_i[k], sl[k]) end_do;
(6)     if rn ≥ r_rn_i then rec_from_i[rn] ← rec_from_i[rn] ∪ {j}
(7)     end_if

(8) when (timer_i has expired) ∧ (|rec_from_i[r_rn_i]| ≥ n − t):
(9)     let suspects = Π \ 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≤j≤n});
(12)    r_rn_i ← r_rn_i + 1

(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

(19) when leader() is invoked by the upper layer:
(20)    let ℓ such that (susp_level_i[ℓ], ℓ) = min({(susp_level_i[j], j)}_{1≤j≤n});
(21)    return (ℓ)
```

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

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.*

Proof The proof is the same as the proof of Lemma 4. 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 round numbers. □ $_{Lemma\ 5}$

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 *B is bounded.*

Proof This lemma follows directly from the fact that no entry of $susp\_level_i[1..n]$ ever decreases and Lemma 5. □$_{Lemma\ 6}$

Lemma 7 *Let p_i be a correct process and p_j a faulty process. Eventually, $susp\_level_i[j] > B$.*

Proof The proof is a simple combination of arguments used in the proofs of the Lemmas 1 and 3.

- As in the proof of Lemma 1, there is a round number rn, such that for any $rn' > rn$, each correct process receives at least $(n - t)$ SUSPICION$(rn', \{\ldots, j, \ldots\}$ messages, from which it follows that the test of Line 16 is always satisfied from $rn + 1$.

- As in the proof of Lemma 3, there is a round number from which the predicate of Line "*" is always satisfied.

It follows that there is a round number from which both the predicates of Line 16 and Line "*" are always satisfied.

Let us now consider a time after which $\min(\{susp\_level_i[x]\}_{1\leq x\leq n}) = B$ (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. $\square_{Lemma\ 7}$

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

Proof It follows from the gossiping mechanism and Lemma 6 that there is a time after which there is a process p_ℓ such that for all the non-crashed processes $susp\_level_i[\ell] = B$. Moreover due to Lemma 7, 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. $\square_{Theorem\ 3}$

Lemma 8 *For any* p_i*, the relation* $\max(\{susp\_level_i[x]\}_{1\leq x\leq n}) - \min(\{susp\_level_i[x]\}_{1\leq x\leq n}) \leq 1$ *is always satisfied.*

Proof [7] Let us first remind that each set of statements is executed atomically, which means here that the execution of Line 5 on one side, and the execution of Line "**" plus Line 17 on the other side, are not interleaved.

Let $INV(sl)$ be the predicate $\max(\{sl[x]\}_{1\leq x\leq n}) - \min(\{sl[x]\}_{1\leq x\leq n}) \leq 1$, 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.

- $INV(susp\_level_i)$ is initially true (all the entries of $susp\_level_i[1..n]$ are initially equal to 0).
- Update of $susp\_level_i[1..n]$ at Line 17.
  Due to the test of Line "**", $INV(susp\_level_i)$ is trivially maintained when p_i executes Line 17.
- Update of $susp\_level_i[1..n]$ at Line 5.
  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 $INV(sl2)$ 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:
    * $\max(a, b) = \max(a + 1, b) = b$.
    * $\max(a, b + 1) = \max(a + 1, b + 1) = b + 1$.

$\square_{Lemma\ 8}$

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 Lemma 5, p_ℓ is a correct process. Moreover, due to the gossiping mechanism, there is a time after which all the p_j that have not crashed are such that $\min(\{susp\_level_j[x]\}_{1\leq x\leq n}\} = B$. The theorem then follows from Lemma 8. $\square_{Theorem\ 4}$

[7] 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.

7 Further weakening the system model

In this section the assumption $\mathcal{A}$ defined above is further weakened. The new assumption, denoted $\mathcal{A}^{-}_{f,g}$, allows the delays experienced by timely messages or the round number distance between the appearance of t-stars to grow unbounded. This model extension 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 functions $f()$ and $g()$

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

  The motivation for the function $f()$ is to "weaken" the constraint $s_{k+1} - s_k \leq D$ used to define the infinite sequence $S = s_1, s_2, \ldots$ that appears in the statement of the assumption $\mathcal{A}$. This constraint can be made specific to each round number, reformulating it as follows: $\forall k \geq 1 : \ s_{k+1} - s_k \leq D + f(s_k)$, thereby allowing the appearance period between consecutive t-stars to grow without bound.

  It is easy to see that the particular function $\forall rn : \ f(rn) = 0$ corresponds to the basic constraint used in the assumption $\mathcal{A}$. The particular function $\forall rn : \ f(rn) = 1 - D$ corresponds to the assumption $\mathcal{A}^{+}$.

- $g()$ is a function from the set of round numbers into the time domain. More precisely, $g(rn)$ 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 4 *A message* ALIVE(rn) *is* (δ, g)-timely *if it is received by its destination process at most* $\delta + g(rn)$ *time units after it has been sent.*

  If $\forall \ rn : \ g(rn) = 0$, 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 $s_{k+1} - s_k \leq D$ is replaced by $s_{k+1} - s_k \leq D + f(s_k)$ in the definition of the sequence S, and (2) for all the ALIVE() message, the notion of δ-timely message is replaced by the notion of (δ, g)-timely message.

An $\mathcal{A}^{-}_{f,g}$-based algorithm Interestingly, if the processes know the functions $f()$ and $g()$, a very simple modification of the $\mathcal{A}$-based algorithm described in Figure 3 provides an $\mathcal{A}^{-}_{f,g}$-based algorithm. These modifications are the following (the new parts are underlined):

- The timer resetting (Line 11) has now to take into account the function $g()$ applied to the next round number. Hence, Line 11 becomes: **set** $timer_i$ **to** $\max(\{susp\_level_i[j]\}_{1 \leq j \leq n}) \underline{+ g(r\_rn_i + 1)}$.
- The definition of the interval used in the test of Line "*" has to take into account the new constraint $s_{k+1} - s_k \leq D + f(s_k)$. This interval is now: $\forall x : rn - (susp\_level_i[k] \underline{+ f(rn)}) < x < rn$.

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

8 Conclusion

This paper has first proposed a weak system model in which an eventual leader can be elected. This model is the classical asynchronous model with process crashes and reliable communication, enriched with what we call the *intermittent rotating t-star* assumption. That assumption states the existence of a process p (the center of the star) and

logical times such that, for a subset of these logical times rn, there are sets $Q(rn)$ of t processes and each process of each $Q(rn)$ receives from p a message tagged rn in a timely manner or among the first $(n-t)$ messages tagged rn it ever receives. 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.

The paper has also presented an algorithm based on that *intermittent rotating t-star* assumption. The presentation has voluntarily been done in a methodological and incremental way. That algorithm enjoys 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 [20], 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 round 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 [3, 4] with this paper we obtain the following theorem:

Theorem 5 *The consensus problem 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. and Toueg S., On Implementing Omega with Weak Reliability and Synchrony Assumptions. *22th ACM Symposium on Principles of Distributed Computing (PODC'03)*, ACM Press, pp. 306-314, 2003.

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

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

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

[5] 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)*, IEEE Computer Society Press, pp. 166-175, Philadelphia (PA), 2006.

[6] 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.

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

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

[9] 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.

[10] Hutle M., Malkhi D., Schmid U. and Zhou L., Chasing the weakest system model for implementing Ω and consensus. *Brief Annoucement, Proc. 8th Int'l Symposium on Stabilization, Safety and Security in Distributed Systems (SSS'06)*, Springer-Verlag LNCS, Dallas (TX), 2006. (Full version in *Tech Report 74/2005*, Institut für Technische Informatik, Technische Universität, Wien (Austria), 2006.)

[11] Jiménez E., Arévalo S. and Fernández A., Implementing unreliable failure detectors with unknown membership. *Information Processing Letters*, 100(2):60-63, 2006.

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

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

Irisa

[14] 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)*, IEEE Computer Society Press, pp. 52-60, 2000.

[15] 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.

[16] Mostéfaoui A., Mourgaya E., and Raynal M., Asynchronous Implementation of Failure Detectors. *Proc. Int'l IEEE Conference on Dependable Systems and Networks (DSN'03)*, IEEE Computer Society Press, pp. 351-360, 2003.

[17] Mostéfaoui A. and Raynal M., Leader-Based Consensus. *Parallel Processing Letters*, 11(1):95-107, 20.0.

[18] Mostéfaoui A., Raynal M. and Travers C., Crash-resilient Time-free Eventual Leadership. *Proc. 23th Int'l IEEE Symposium on Reliable Distributed Systems (SRDS'04)*, IEEE Computer Society Press, pp. 208-217, 2004.

[19] 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.

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

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