From W to Omega: a Simple Bounded Quiescent Reliable broadcast-based Transformation

Achour Mostefaoui, Sergio Rajsbaum, Michel Raynal, Corentin Travers

▶ To cite this version:

Achour Mostefaoui, Sergio Rajsbaum, Michel Raynal, Corentin Travers. From W to Omega: a Simple Bounded Quiescent Reliable broadcast-based Transformation. [Research Report] PI 1759, 2005, pp.9. inria-00000663

HAL Id: inria-00000663
https://inria.hal.science/inria-00000663

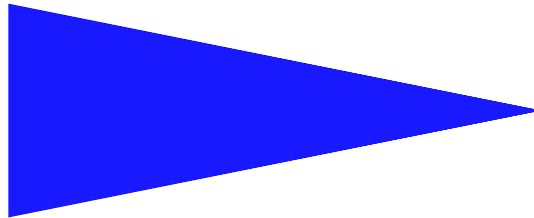
Submitted on 14 Nov 2005

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

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

IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1759



FROM $\diamond\mathcal{W}$ TO Ω: A SIMPLE BOUNDED QUIESCENT RELIABLE BROADCAST-BASED TRANSFORMATION

A. MOSTEFAUI S. RAJSBAUM M. RAYNAL C. TRAVERS

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

From $\diamond\mathcal{W}$ to Ω: a Simple Bounded Quiescent Reliable broadcast-based Transformation

A. Mostefaui* S. Rajsbaum** M. Raynal*** C. Travers****

Systèmes communicants

Publication interne n˚ 1759 — Novembre 2005 — 9 pages

Abstract: $\diamond\mathcal{W}$ is the class of failure detectors that ensure that every crashed process is eventually suspected by a correct process, and eventually there is correct process that is never suspected. Ω is the class of failure detectors that ensure that eventually all the processes trust the same correct process. This paper presents two algorithms that transform any failure detector of $\diamond\mathcal{W}$ into a failure detector of the class Ω. Based on an underlying reliable broadcast facility, these algorithms are quiescent and use message whose size is bounded ($\log_2(n)$ bits and $2\log_2(n)$, respectively, where n is the number of processes). When we consider the additional cost of implementing the underlying reliable broadcast, they still compare favorably with existing algorithms. From a methodology point of view, it is worth noticing that the use of a reliable broadcast (i.e., a modular approach) allows designing simple and efficient transformations.

Key-words: Asynchronous system, Distributed algorithm, Eventual leader, Failure detector, Fault-tolerance, Layer abstraction, Process crash, Reliable broadcast.

(Résumé : tsvp)

* IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France achour@irisa.fr
** Instituto de Matemáticas, UNAM, D. F. 04510, Mexico rajsbaum@matem.unam.mx
*** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, raynal@irisa.fr
**** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, travers@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

De la classe de détecteurs de fautes $\diamond\mathcal{W}$ à la classe Ω

Résumé : Ce rapport présente un protocole simple qui construit un détecteur de fautes de la classe Omega à partir de n'importe quel détecteur de fautes de la classe $\diamond\mathcal{W}$.

Mots clés : Systèmes répartis asynchrones, Tolérance aux fautes, Crash de processus, Diffusion fiable, Détecteur de fautes, Leader, Réduction.

1 Introduction

Consensus is one of the most fundamental problem in fault-tolerant distributed computing: each process proposes a value, and every non-faulty process must decide a value (termination) such that no two different values are decided (agreement) and the decided value is a proposed value (validity). Despite the simplicity of its definition and its use as building block to solve distributed agreement problems, consensus cannot be solved in asynchronous system where even a single process can crash [6].

The *failure detector* approach [3] is one of the efforts developed to circumvent this impossibility result. A failure detector can be seen as a distributed oracle made up of a set of modules, each associated with a process. The module attached to process p provides p with information related to failures. Failure detectors are divided into *classes* based on the particular type of information they provide on failures. One can identify two main characteristics of the failure detector approach:
- As a failure detector class is defined by a set of *abstract* properties, a failure detector-based protocol relies only on the properties that define the failure detector, regardless of the way they are implemented in a given system. This *software engineering* dimension of the failure detector approach favors protocol design, modularity and transportability.
- Given a distributed computing problem P that cannot be solved in the presence of process failures, the failure detector approach allows investigating and stating the minimal assumptions that allow solving P [5]. This is the *computability* dimension of the failure detector approach.

When we consider the crash failure model, is process is *correct* in a run if it does not crash in that run; otherwise it is *faulty*. Considering that failure model, this paper focuses on the classes of failure detectors denoted $\diamond\mathcal{W}$ and Ω [2, 3]:

- The class $\diamond\mathcal{W}$ includes all the failure detectors that satisfies the following properties:
  - Weak completeness: Eventually every process that crashes is permanently suspected by some correct process (notice that two crashed processes can be suspected by two different correct processes).
  - Eventual weak accuracy: Eventually there is a correct process that is not suspected by any correct process.
- The class Ω includes all the failure detectors that satisfies the following Eventual leadership property: Eventually, all the correct processes trust the same process (i.e., are provided with the same leader) that is a correct process.

When we consider the consensus problem, the class $\diamond\mathcal{W}$ is fundamental because it has been shown to the weakest that allows solving it in asynchronous distributed systems where a majority of processes are correct [2]. To attain this goal, Chandra, Hadzilacos and Toueg have introduced the class Ω as an intermediate step in their proof, and shown that $\diamond\mathcal{W}$ and Ω are equivalent (i.e., any failure detector of one of these classes can be transformed into a failure detector of the other class).

The transformation from Ω to $\diamond\mathcal{W}$ is trivial: a process suspects all the processes but its current leader. The transformation from $\diamond\mathcal{W}$ to Ω presented in [2], aimed at proving a minimality result, is complicated and inefficient (the price to be paid in the context of generality when looking for a minimality result). To our knowledge, the only other paper describing transformations from $\diamond\mathcal{W}$ to Ω is [4] where Chu presents two transformations. The first requires each message to carry an array of counters, some of them growing indefinitely. Moreover, it is not quiescent (a distributed algorithm is *quiescent* if, in every run, the processes eventually stop sending messages). In the second transformation each message carries a sequence number plus a set of process identities; as this transformation is quiescent, the sequence numbers stops increasing and consequently, the size of each message remains finite in any run (i.e., even in an infinite run).

The present paper presents a transformation (and an improvement) from $\diamond\mathcal{W}$ to Ω for asynchronous message-passing systems equipped with a reliable broadcast communication primitive. The transformation, that is incredibly simple, is quiescent and requires each message to carry only one process identity. It follows that $\log_2(n)$ (where n is the total number of processes) is a bound on its message size (measured in bits). Its improved version allows discarding obsolete messages, thus allowing the saving of message sending and processing time, with the possibility to converge quicker to the eventual common leader. To attain this goal, each process uses an additional local integer matrix whose entries stop increasing after some time. Moreover,

each message is required to carry an additional process identity (namely the identity of its sender). The size of the messages used by this second transformation is consequently upper bounded by $2\log_2(n)$ bits.

2 Computation Model

2.1 Process Model

The process model has been sketched in the introduction. It consists of a finite set Π of $n \geq 2$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. Each process knows Π. There is no assumption about the relative speed of processes: they are asynchronous.

A process can fail by *crashing*, i.e., by prematurely halting. It behaves correctly (i.e., according to its specification) until it (possibly) crashes. As already indicated, a process is *correct* in a run if it does not crash in that run, otherwise it is *faulty*. (Let t denote the maximum number of processes that can crash in a run, $1 \leq t < n$. The transformations described in the paper are independent from t.)

2.2 Communication Model

The communication system provides the processes with a *reliable broadcast* abstraction [7]. Such an abstraction is made up of two primitives *Broadcast()* that allow a process to send messages, and *Deliver()* that allows a process to receive messages. We say accordingly that a message is broadcast or delivered by a process. The behavior of these primitives is defined the following properties:

- Validity: If a process delivers m, then some process has broadcast m. (No spurious messages.)
- Integrity: A process delivers a message m at most once. (No duplication.)
- Termination: If a correct process broadcasts or delivers a message m, then all the correct processes delivers m. (No message broadcast or delivered by a correct process is missed by a correct process.)

As we can see, the messages broadcast by a process are not necessarily delivered in their sending order. Moreover, different processes can be delivered these messages in different order. There is no assumption on message transfer delays. The communication system is consequently reliable and asynchronous.

Differently from the $\diamond\mathcal{W}$ and Ω abstractions that cannot be built on top of an asynchronous distributed system (as they allow solving consensus, their implementation would contradict the consensus impossibility result [6]), the reliable broadcast abstraction can be implemented from point-to-point send/receive primitives. Moreover such implementations are quiescent (see also the discussion in Section 5).

3 From $\diamond\mathcal{W}$ to Ω: a Basic Transformation

3.1 The Transformation

The first algorithm transforming any failure detector of the class $\diamond\mathcal{W}$ into a failure detector of the class Ω is described in Figure 1. It is made up of two tasks containing very few statements.

The task $T1$ is an infinite loop that generates messages according to the current value of a local predicate. The task $T2$ is associated with message delivery. When it is delivered a message, this task uses only the content of the message as defined by its sender (namely, a process identity). The additional control information possibly used by the underlying layer to implement the reliable broadcast abstraction remains hidden to $T2$. This layer independence is in perfect agreement with the modularity approach as it makes the transformation algortihm dependent only on the abstract properties that define the reliable broadcast abstraction, thereby making it independent from the particular way this abstraction is implemented in a given context. So, the algorithm is independent from the way both the underlying failure detector of the class $\diamond\mathcal{W}$ and the reliable broadcast are implemented. It is based only on their properties.

Local variables The local variable $leader_i$ (initialized to 1) is the output of the transformation. It provides the upper layer with the identity of the current leader, as far as p_i is concerned. As this variable can be concurrently accessed by the tasks defining the process p_i ($T1$, $T2$ and the task associated with the upper layer), it is assumed to be atomic.

The local variable $suspected_i$ is provided by the underlying failure detector module of the class $\diamond\mathcal{W}$. At any time, it contains the set of the identities of the processes that are currently suspected by p_i (p_i can only read $suspected_i$). The set of set variables $(suspected_i)_{1\leq i\leq n}$ satisfies the weak completeness and eventual weak accuracy properties defined in the introduction.

The function $\mathsf{Next}(x)$ is used to denote the identity y of the process that follows p_x on the logical directed ring $p_1, p_2, \ldots, p_n, p_1$, i.e., $y = (x \mod n) + 1$.

```
Init: leader_i ← 1

Task T1 is repeat forever
              if (leader_i ∈ suspected_i) then Broadcast GO_TO_NEXT(leader_i) end_if
           end_repeat

Task T2 is repeat forever
              wait until (a new message GO_TO_NEXT(ℓ) such that (leader_i = ℓ) is delivered);
              leader_i ← Next(leader_i)
           end_repeat
```

Figure 1: From $\diamond\mathcal{W}$ to Ω, basic transformation

Process behavior The principle that underlies the algorithm is very simple. As the eventual permanent leader has to be a correct process, the processes strive to elect a process that, after some time, is no longer suspected. In order not to miss such a process, they consider they are all placed on a logical directed ring (as defined above) and start considering p_1 as the leader (initialization).

If p_i suspects its current leader $leader_i = \ell$, it broadcasts a GO_TO_NEXT(ℓ) message. When a process p_j receives such a message, it consumes it and moves $leader_j$ one step ahead along the ring if $leader_j = \ell$. If $leader_j \neq \ell$, it does not consume the message (keeping it for the next time it will have $leader_j = \ell$). This means that a process p_j accepts the suspicion of p_ℓ issued by p_i only when it considers p_ℓ as its current leader. It follows from this simple mechanism that the successive values of the variable $leader_i$ of any process p_i are 1, then 2, etc., then n, 1,... Logically, the $leader_i$ variables advance "synchronously" along the ring, their progress being triggered by the GO_TO_NEXT(ℓ) messages sent by the processes[1], each GO_TO_NEXT(ℓ) message making each $leader_i$ variable progress from ℓ to $\mathsf{Next}(\ell)$.

It is important to notice that, when a message GO_TO_NEXT($\ell1$) sent by a process p_i and a message GO_TO_NEXT($\ell2$) sent by a process p_j (with possibly $p_i = p_j$) are such that $\ell1 = \ell2$, they are two messages with the same content and consequently cannot be distinguished by a destination process p_k (the identity of the sender of a message is not part of the message and cannot consequently be used when the message is delivered). This means that the processing of a message GO_TO_NEXT(ℓ) and the processing of another message carrying the same content are executed in the same context and produce the same result: both allows $leader_k$ to proceed from the value ℓ to $\mathsf{Next}(\ell)$. This means that between the processing of two such messages, $leader_k$ has taken the successive values: $\mathsf{Next}(\ell)$, $\mathsf{Next}(\mathsf{Next}(\ell))$,..., until ℓ, i.e., a full turn around the logical directed ring. The proof shows that this simple mechanism allows the set of $leader_i$ variables to eventually contain forever the same value that is the identity of a correct process.

3.2 Correctness Proof

The proof considers an arbitrary run. Let C denote the set of processes that are correct in that run.

[1] This principle is close to a basic mechanism provided by synchronizers used to globally synchronize a network [1].

Lemma 1 $\forall i \in C : \exists\ \lambda_i \in \Pi$ *and a time* τ_i *such that* $\forall \tau_i' \geq \tau_i : leader_i = \lambda_i$.

Proof Let us first make the two following observations:
- O1: Due the eventual weak accuracy property of $\diamond\mathcal{W}$, there is a process identity $\ell \in C$ and a time τ such that p_ℓ is never suspected after τ, which means that, after that time, no local predicate $\ell \in suspected_k$ can be satisfied. It follows that the number of messages GO_TO_NEXT(ℓ) that are sent is finite.
- O2: Due to the fact that the process identities are arranged along a logical ring, the sequence of the successive values of the variable $leader_i$ is 1, 2, ..., n, 1, etc.

Let us assume (by way of contradiction) that there is no process identity λ_i such that after some time $leader_i = \lambda_i$ remains true forever. If follows from O2 that $leader_i$ takes each value k, $1 \leq k \leq n$, infinitely often, and consequently p_i executes $leader_i \leftarrow \mathsf{Next}(\ell)$ infinitely often. But this contradicts O1 as the number of GO_TO_NEXT(ℓ) that are received by p_i is finite. $\Box_{Lemma\ 1}$

The next corollary follows directly from the previous lemma and the fact that a crashed process does not send message.

Corollary 1 $\forall i, \ell$: *the number of* GO_TO_NEXT(ℓ) *sent by* p_i *is finite.*

Lemma 2 $\forall i, j \in C : \lambda_i = \lambda_j$. (*In the following* λ *denotes that value.*)

Proof Due to the reliable broadcast, p_i and p_j delivers the same bag of messages[2]. If follows from corollary 1 that these bags are finite. Due to the fact that p_i and p_j consume the messages according to the same ring order, and the fact that the bags are equal and finite, it follows that $\lambda_i = \lambda_j$. $\Box_{Lemma\ 2}$

Lemma 3 λ *is the identity of a correct process.*

Proof Let us assume (by contradiction) that λ is the identity of a faulty process. Due to the weak completeness property of $\diamond\mathcal{W}$, there is a correct process p_c that suspects permanently p_λ after some time $\tau 1$. On another side, it follows from Lemma 1 that the predicate $leader_c = \lambda$ remains permanently true from some time τ_c. It follows that there is a time $\geq \max(\tau_c, \tau 1)$ at which p_c broadcasts a message GO_TO_NEXT(λ). When p_c delivers this message it executes $leader_c \leftarrow \mathsf{Next}(\lambda)$, thereby contradicting Lemma 1. $\Box_{Lemma\ 3}$

Theorem 1 *The algorithm described in Figure 1 is a quiescent transformation from* $\diamond\mathcal{W}$ *to* Ω. *Moreover, the size of each message is* $\log_2(n)$ *bits.*

Proof Due to the Lemmas 1 and 2, there is a time after which all the correct processes have the same leader. Due to Lemma 3, this leader is a correct process. The quiescence of the transformation follows from Corollary 1. Finally, each message carries one process identity. $\Box_{Theorem\ 1}$

4 From $\diamond\mathcal{W}$ to Ω: an Improvement

4.1 Discarding Messages

Considering the transformation described in Figure 1, let us analyze the following scenario where, while each process p_i is such that $leader_i = \ell$, p_ℓ crashes and the predicate $\ell \in suspected_i$ is satisfied before p_i receives a message GO_TO_NEXT(ℓ). Every process p_i consequently broadcasts a message GO_TO_NEXT(ℓ). As we have seen in Section 3.1, until a new leader is elected, each of these messages is taken into account by the task $T2$ of each process p_i during a turn of the ring done by its $leader_i$ variable (as we have seen, such a message, taken into account when $leader_i = \ell$, triggers the update $leader_i \leftarrow \mathsf{Next}(\ell)$). This means that the

[2] A bag is a set that can contain several copies of the same element; here, an element of the bag is a message. From the bag pont of view, a message GO_TO_NEXT(k) sent by p_x and a message GO_TO_NEXT(k) sent by p_y are two copies of the same element.

concurrent suspicions of p_ℓ, each generating a GO_TO_NEXT(ℓ) message, are processed sequentially by each process p_i (each "during a turn of $leader_i$ along the ring"). The idea of the improvement is to allow the algorithm to consider all these messages as a single message, thereby generating a single advance of a $leader_i$ variable from ℓ to $\mathsf{Next}(\ell)$, saving accordingly message buffer space.

To attain this goal, for any pair (x, ℓ), the set including the xth GO_TO_NEXT(ℓ) message received from each process p_k is considered by its receiving process p_i as a single GO_TO_NEXT(ℓ) message; p_i considers then the first of these messages and discards the others (if any). This requires (1) to associate the identity of its sender with each message, and (2) to provide each process p_i with additional local variables that allow it to decide which messages have to be processed and which ones can be discarded. These local data structures are a bag and an integer matrix.

- bag_i is a bag where p_i stores the messages GO_TO_NEXT(ℓ) it has to process, one each time the predicate $leader_i = \ell$ becomes satisfied. (The primitive "Insert a into bag_i" adds a copy of a into bag_i, while the primitive "Extract a from bag_i" suppresses a copy of a from bag_i.)
- $M_i[1:n, 1:n]$ is an integer matrix whose meaning is the following: $M_i[k, \ell]$ is the number of GO_TO_NEXT(ℓ) messages that have been broadcast by p_k, to p_i knowledge.

  Let $y = \max(\{M_i[j, \ell]\}_{1 \le j \le n})$ just before p_i delivers a message GO_TO_NEXT(ℓ) from p_k. Just after it has received the message, $M_i[k, \ell] = x$ means that this is the xth GO_TO_NEXT(ℓ) message that p_i delivers from p_k. According to the previous discussion, p_i saves that message in bag_i for future processing if $(M_i[k, \ell] =)\ x > y$; otherwise p_i discards the message.

The corresponding transformation algorithm is described in Figure 2. It is made up of three tasks. The tasks $T1$ is the same in both transformations. The task $T2$ of the first transformation is now split in two tasks $T2'$ and $T2''$ in the second transformation. The size of a message is $2\log_2(n)$ as a message has now to carry the identity of its sender. As we will see in the proof, as the previous one, this transformation is quiescent. It follows that the values contained in any matrix M_i are finite even in infinite runs.

```
Init: leader_i ← 1

Task T1   is repeat forever
                if (leader_i ∈ suspected_i) then Broadcast GO_TO_NEXT (leader_i) end_if
             end_repeat

Task T2'  is repeat forever
                wait until (a new message GO_TO_NEXT(ℓ) is delivered);
                let p_k = sender of the message;
                M_i[k,ℓ] ← M_i[k,ℓ] + 1;
                if (∀j ≠ k : M_i[k,ℓ] > M_i[j,ℓ]) then Insert GO_TO_NEXT(ℓ) into bag_i
                                                  else Discard the message      end_if
             end_repeat

Task T2'' is repeat forever
                wait until (∃ GO_TO_NEXT(ℓ) ∈ bag_i such that (leader_i = ℓ));
                Extract GO_TO_NEXT(ℓ) from bag_i; leader_i ← Next(leader_i)
             end_repeat
```

Figure 2: From $\diamond\mathcal{W}$ to Ω, improved transformation

4.2 Correctness Proof

The statements of Lemma 1, Lemma 2, Lemma 3, Corollary 1 and Theorem 1 remain valid when we consider the transformation described in Figure 2. As far Lemma 1 is concerned, its proof is the same as it relies only on the ring structure and the eventual weak accuracy property of $\diamond\mathcal{W}$. (Corollary 1 remains consequently

valid.) Except for the message size (that is now $2\log_2(n)$ bits), the proof of Theorem 1 is the same as before. Only the proofs of Lemma 1 and Lemma 2 have to be modified to take into account the new features of the transformation.

Proof (New proof for Lemma 2.)
Let BAG_i be the abstract bag of all the messages that p_i inserts into its concrete bag bag_i. To show that $\forall i,j \in C: \lambda_i = \lambda_i$, we show that BAG_i and BAG_j are finite and equal. (The rest of the proof is then same as in the initial proof.)

Among the deliveries to p_i of the GO_TO_NEXT(ℓ) messages broadcast by p_k, let us consider the xth delivery. We have $M_i[k,\ell] = x$ just after the delivery to p_i of the corresponding message; p_i adds this message to bag_i if and only if $(\forall j \neq k:\ x > M_i[j,\ell])$ (otherwise it discards it). This means that, when we consider the set made up of the xth GO_TO_NEXT(ℓ) message delivered from p_1 (if any), the xth GO_TO_NEXT(ℓ) message delivered from p_2 (if any), etc., the xth GO_TO_NEXT(ℓ) message delivered from p_n (if any), p_i considers for future processing only the first of them it delivers. Due to Corollary 1, BAG_i is finite. As p_j does the same, and (due to the validity, integrity and termination properties of the reliable broadcast) p_i and p_j are delivered the same finite bag of messages, it follows that BAG_i and BAG_j are finite and equal. $\square_{Lemma\ 2}$

Proof (New proof for Lemma 3.) We have to show that λ is the identity of a correct process.
Due to Corollary 1, each entry of a matrix M_i stops increasing. If λ is the identity of a faulty process, there is a correct process p_c such that eventually $\lambda \in suspected_c$ remains forever true. Due to Lemmas 1 and 2, there is a time after which $leader_c = \lambda$ is always true. It follows that, after some finite time, infinitely often p_c broadcasts GO_TO_NEXT(λ). Consequently the entry $M_i[c,\lambda]$ of a correct process never stops increasing. A contradiction with the fact that each entry of a matrix M_i stops increasing. $\square_{Lemma\ 3}$

5 Concluding Remark

Both transformations are quiescent and use simple mechanisms to implement an eventual leader facility of the class Ω from any failure detector of the class $\diamond\mathcal{W}$. Both assume that the underlying asynchronous system provides the processes with a built-in reliable broadcast facility. In such a context, we have seen that each message has a bounded size (namely $\log_2(n)$ bits for the first transformation, and $2\log_2(n)$ for the second). The proposed transformations clearly demonstrate that a message needs to carry no more than one or two process identities as soon as a reliable broadcast can be used.

At the underlying network level where the reliable broadcast abstraction is implemented, the size of a message becomes the size of its content (as defined by the upper layer), plus the size of the control information (if needed) it has to carry in order to implement the reliable broadcast abstraction. Implementing a reliable broadcast can require associating a proper identity with each message [7]. This can be done by associating a pair <sequence number, sender identity> with each message[3]. Whatever the transformation, at the network level, a message has then to carry a sequence number plus two process identities. As the proposed transformations are quiescent and there are reliable broadcast protocols that are quiescent, this means that the sequence numbers used by each process remain finite in any (possibly infinite) run. Consequently, in any run, the amount of memory space and the size of each message are finite. The second transformation from $\diamond\mathcal{W}$ to Ω proposed by Chu [4] enjoys the same property, but while his transformation requires each message to carry a sequence number plus a set of identities of suspected processes (such a set can contain up to n identities), seen from the network level, our transformations require each message to carry a sequence number plus only two process identities. Finally, in the proposed transformations, a process issues a reliable broadcast only when its current leader is suspected. Moreover, these transformations are stable in the sense that a common leader can be demoted only when its is suspected. Differently, in [4], a process issues a broadcast when the set of processes it suspects increases or when it generates a higher sequence number. It follows that, when we consider the proposed transformations and the transformations described in [4] at

[3]It is important to remark that such a pair has to be associated with each message when one has to implement a reliable channel on top of a fair lossy unreliable channel, which means that implementing reliable broadcast on top of a point-to-point communication network and implementing reliable channels have the same cost from a message size point of view.

the send/receive network level, they constitute two distinct approaches to go from $\diamond\mathcal{W}$ to Ω. The paper has investigated the benefits provided by an underlying reliable broadcast facility (design simplicity and efficiency measured from a message size criterion).

References

[1] Awerbuch B., Complexity of Network Synchronization. *Journal of the ACM*, 32(4):804-823, 1985.

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

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

[4] Chu F., Reducing Ω to $\diamond\mathcal{W}$. *Information Processing Letters*, 76(6):293-298, 1998.

[5] Delporte-Gallet C., Fauconnier H., Guerraoui R., Hadzilacos V., Kouznetsov P. and Toueg S., The Weakest Failure Detectors to solve Certain Fundamental Problems in Distributed Computing. *Proc. 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, pp. 338-346, 2004.

[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] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press, New-York, pp. 97-145, 1993.