A Versatile Family of Consensus Protocols Based on Chandra-Toueg's Unreliable Failure Detectors

Michel Hurfin, Achour Mostéfaoui, and Michel Raynal

Abstract—This paper is on *consensus protocols* for asynchronous distributed systems prone to process crashes, but equipped with Chandra-Toueg's unreliable failure detectors. It presents a unifying approach based on two orthogonal versatility dimensions. The first concerns the class of the underlying failure detector. An instantiation can consider any failure detector of the class $\mathcal{S}$ (provided that at least one process does not crash), or $\diamond\mathcal{S}$ (provided that a majority of processes do not crash). The second versatility dimension concerns the message exchange pattern used during each round of the protocol. This pattern (and, consequently, the round message cost) can be defined for each round separately, varying from $O(n)$ (centralized pattern) to $O(n^2)$ (fully distributed pattern), n being the number of processes. The resulting versatile protocol has nice features and actually gives rise to a large and well-identified family of failure detector-based consensus protocols. Interestingly, this family includes at once new protocols and some well-known protocols (e.g., Chandra-Toueg's $\diamond\mathcal{S}$-based protocol). The approach is also interesting from a methodological point of view. It provides a precise characterization of the two sets of processes that, during a round, have to receive messages for a decision to be taken (liveness) and for a single value to be decided (safety), respectively. Interestingly, the versatility of the protocol is not restricted to failure detectors: a simple timer-based instance provides a consensus protocol suited to partially synchronous systems.

Index Terms—Asynchronous distributed system, consensus problem, consensus protocols, crash failure, fault-tolerance, quorum, unreliable failure detector.

1 Introduction

Understanding the deep structure and the basic design principles of protocols solving fundamental distributed computing problems is an important and challenging task. Such efforts have been produced for basic problems such as distributed mutual exclusion [10], [22], distributed deadlock detection [3], [13], and a few others. When we consider the consensus problem in asynchronous distributed systems prone to process crashes, several failure detector-based protocols have been designed. But, no unifying approach that reveals their deep structure and their common algorithmic principles has yet been proposed. This paper is an endeavor to produce such a unifying conceptual framework that gives an account of a large class of failure detector-based consensus protocols.

1.1 Context of the Study

Informally, the consensus problem is defined in the following way: Each process proposes a value and all noncrashed processes have to agree on a common value which has to be one of the proposed values. Solving this problem in asynchronous distributed systems where processes can crash is far from being a trivial task. More precisely, it has been shown by Fischer et al. [7] that there is no (deterministic) solution to this problem if processes (even only one) may crash. A major advance proposed so far to circumvent this impossibility result lies in the *unreliable failure detector* concept, proposed and investigated by Chandra and Toueg [4] and Chandra et al. [5]. A failure detector is basically defined by two properties: A *completeness* property that is on the actual detection of failures and an *accuracy* property that limits the mistakes a failure detector can make. Chandra and Toueg defined two completeness properties and four accuracy properties that allowed them to define eight classes of failure detectors. In this paper, we are interested in asynchronous distributed systems equipped with a failure detector of the class $\mathcal{S}$ or with a failure detector of the class $\diamond\mathcal{S}$. Both classes are characterized by the same completeness property, namely, "Eventually, every crashed process is suspected by every correct process." They are also characterized by the same basic accuracy property, namely, "There is a correct process that is never suspected." But these two classes differ in the way (modality) their failure detectors satisfy this basic accuracy property. More precisely, the failure detectors of $\mathcal{S}$ *perpetually* satisfy the basic accuracy property, while the failure detectors of $\diamond\mathcal{S}$ are allowed to satisfy it only *eventually*. Protocols based on such failure detectors have been designed. They all proceed in asynchronous rounds whose aim is to make the processes eventually converge to a same value (and then decide it). The main features of these protocols are described in the next paragraphs. (A more detailed presentation of their underlying principles in done in Appendix A.)

Several $\mathcal{S}$-based consensus protocols have been designed [4], [8], [18], [25]. The protocols described in [4], [18], [25]

• *The authors are with the Institut de Recherche en Informatique et Systèms Aléatoires, Campus de Beaulieu, 35042 Rennes Cedex, France. E-mail: {hurfin, achour, raynal}@irisa.fr.*

Manuscript received 30 Mar. 2000; revised 13 Dec. 2000; accepted 13 Aug. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111811.

require that each process executes n rounds (where n is the total number of processes). In these protocols a round is made up of a single communication step. During a round of the $\mathcal{S}$-based consensus protocol described in [4], each process broadcasts a message and, hence, a round costs n^2 messages. Differently, in the $\mathcal{S}$-based protocols described in [18], [25], a single process per round broadcasts a message, and, consequently, a round costs n messages. These protocols force the processes to execute n rounds, whatever the number of actual crashes. Differently, the $\mathcal{S}$-based protocol described in [8] allows the processes to decide in less rounds in favorable circumstances, i.e., when there are less than f crashes and no erroneous failure suspicion (where f is the maximal number of processes that may crash). In that sense, this protocol allows for early decision. This is obtained by providing the processes with an appropriate decision predicate. If there are no erroneous suspicions, this protocol requires at most $(f+1)$ rounds. Each round is coordinated by a process and is made up of two communication steps. During the first step, the coordinator broadcasts a message that is answered by each process, hence, a round involves $O(n)$ messages. All these protocols assume $f \leq n-1$ and are consequently optimal with respect to the number of crashes that can be tolerated.

A few $\diamond\mathcal{S}$-based consensus protocols have been proposed [4], [12], [23]. They are based on the rotating coordinator paradigm. During each round, a predetermined process tries to impose its current estimate of the decision value to become the decided value. These protocols mainly differ in the message exchange pattern they use to help the current round coordinator to impose its value: the protocol described in [4] uses a centralized message exchange pattern (and, consequently, a round costs $O(n)$ messages), while the protocols described in [12], [23] use a distributed pattern (and, consequently, a round costs $O(n^2)$ messages). These last two protocols differ in the way they cope with failures and erroneous failure suspicions. Basically, the protocol described in [12] "trusts" the failure detector, while the protocol described in [23] does not (the behaviors of these protocols are compared in [12]). All the previous $\diamond\mathcal{S}$-based consensus protocols assume $f < n/2$, which has been shown to be a necessary condition [4]. So, in asynchronous distributed systems equipped with a $\diamond\mathcal{S}$ failure detector, they are also optimal with respect to the number of crashes that can be tolerated. It is important to note that, as the accuracy property of $\diamond\mathcal{S}$ is only eventual, it is not possible to bound the number of rounds of these protocols.

Although the previous $\mathcal{S}$-based and $\diamond\mathcal{S}$-based consensus protocols share common mechanisms (asynchronous rounds, rotating coordinator, keeping of an estimate value, etc.), they have been designed, developed, and proven independently. A first effort to provide a unified design of failure-detector-based consensus protocols has been presented in [17]. That paper presented a consensus protocol that is generic with respect to the failure detector it can use. More precisely, this protocol can be instantiated with a failure detector that belongs to $\mathcal{S}$ (provided that $f \leq n-1$) or $\diamond\mathcal{S}$ (provided that $f < n/2$). This protocol proceeds in asynchronous rounds. Each round is managed by a predetermined coordinator and is made up of one or two communication steps. Moreover, the round message exchange pattern is distributed and, consequently, involves $O(n^2)$ messages per round. Whatever the failure detector it is instantiated with, this protocol allows for early decision.

1.2 Content of the Paper

The present paper introduces a generic protocol that actually defines a family of failure-detector-based consensus protocols. It actually generalizes the approach introduced in [17]. Like other consensus protocols, the proposed generic protocol is round-based, each round being coordinated by a predetermined coordinator. Its versatility lies in two "orthogonal" dimensions. The first dimension lies in the fact the protocol can be instantiated with any failure detector of $\mathcal{S}$ or $\diamond\mathcal{S}$ (that is the dimension investigated in [17]). Its second versatility dimension concerns the message exchange pattern used during each round. Actually, this pattern and, consequently, the message cost, can be statically or dynamically defined for each round. It is not fixed once for all, but can be tuned for each round separately. It can vary from $O(n)$ (in that case the message exchange pattern is centralized) to $O(n^2)$ (in that case the message exchange pattern is fully distributed). This means it is possible to define a parameter k_r ($1 < k_r \leq n$) associated with each round r, such that the message cost of the round r is $O(k_r n)$.[1]

Another design principle of this family of consensus protocols is to *a priori* trust the information provided by the underlying failure detector. This has an interesting consequence. If the failure detector is tuned to very seldom make mistakes (a frequent case in practice), the resulting consensus protocols are particularly efficient.

The proposed approach shows a deep unity in the design of a family of failure detector-based consensus protocols. This has several advantages. As the approach proposes a single framework giving rise to a family of consensus protocols, a single proof is necessary, whatever the instantiation chosen ($\mathcal{S}$ versus $\diamond\mathcal{S}$ for the failure detector class and centralized versus distributed for the message exchange pattern of each round). This design unity has been made possible by clearly separating the "locking of the decided value" mechanism (see Section 3) from the message exchange pattern. Moreover, in addition to new failure detector-based consensus protocols that are obtained, the design modularity of the approach allows us to derive (and obtain as particular cases) several existing consensus

1. To our knowledge, one of the very first works that systematically considered both centralized and distributed message exchange patterns in the context of agreement problems is [24]. It presents a necessary and sufficient condition for an *Atomic Commit* protocol to be nonblocking and proposes two nonblocking atomic commit protocols (namely, a centralized protocol and a fully distributed protocol). No unifying framework for the message exchange pattern is proposed.

protocols (e.g., the $\mathcal{S}$-based protocols of [8], [17], the $\diamond\mathcal{S}$-based protocols of [4], [17]).[2] It is also interesting to note that the proposed protocol is generic enough to capture rotating coordinator-based consensus protocols that are not failure detector-based. A particularly simple timer-based instance provides a consensus protocol suited to partially synchronous systems [6].

Second, the proposed approach shows that the consensus protocols where the message exchange pattern of each round is centralized and the consensus protocols where this pattern is distributed, are neither conceptually different nor antagonistic. They merely are two "extreme" instantiations of a very same framework. Moreover, as previously indicated, the message exchange pattern can be centralized in some rounds and fully (or partially) distributed in other rounds.

The messages exchanged in $\diamond\mathcal{S}$-based consensus protocols [4], [12], [17], [23] carry their round number, an estimate of the decision value and some additional control information. When we consider the proposed generic protocol, this additional control information is a timestamp (whose value is a round number). It is shown that when the message exchange pattern is fully distributed, this timestamp can be replaced by a Boolean flag. In that case, when the accuracy property is eventual, the only unbounded value a message has to carry is its round number.

1.3 Organization of the Paper

So, as indicated at the beginning of this Introduction, this paper is an effort to understand the deep structure of (a family of) failure detector-based consensus protocols. It is composed of seven sections. Section 2 introduces the computation model and Chandra-Toueg's failure detectors. Then, Section 3 describes the versatile protocol that gives rise to a family of failure detector-based consensus protocols. Section 4 proves its correctness. Section 5 discusses instantiations of the versatile protocol and also shows that several known consensus protocols are particular instances of it. Section 6 discusses some features of the protocol. Finally, Section 7 concludes the paper.

2 Asynchronous Distributed Systems, Failure Detectors, and the Consensus Problem

The system model is patterned after the one described in [4], [7], [14]. A formal introduction to failure detectors is provided in [4].

2.1 Asynchronous Distributed System with Process Crash Failures

We consider a system consisting of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. A process can fail by *crashing*, (i.e., by prematurely halting). It behaves correctly (i.e., according to its specification) until it possibly crashes. By definition, a *correct* process is a process that does not crash. Let f denote the maximum number of processes that can crash ($f \leq n - 1$).

Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are not required to be FIFO. They are assumed to be reliable in the following sense: they do not create, duplicate, alter, or lose messages. This means that a message sent by a process p_i to a process p_j is assumed to be eventually received by p_j, if p_j is correct.[3]

The multiplicity of processes and the message-passing communication make the system *distributed*. There is no assumption about the relative speed of processes or the message transfer delays. This absence of timing assumptions makes the distributed system *asynchronous*.

2.2 Unreliable Failure Detectors

Informally, a failure detector consists of a set of modules, each one attached to a process: the module attached to p_i maintains a set (named $suspected_i$) of the identities of the processes it currently suspects to have crashed. Any failure detector module is inherently unreliable: it can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Moreover, suspicions are not necessarily stable: a process p_j can be added to or removed from a set $suspected_i$ according to whether p_i's failure detector module currently suspects p_j or not. As in papers devoted to failure detectors, we say "process p_i suspects process p_j" at some time, if at that time we have $j \in suspected_i$.

As indicated in the introduction, a failure detector class is formally defined by two abstract properties, namely, a *Completeness* property and an *Accuracy* property. In this paper, we consider the following completeness property [4]:

- **Strong Completeness**. Eventually, every process that crashes is permanently suspected by every correct process.

Among the accuracy properties defined by Chandra and Toueg [4] we consider here the two following:

- **Perpetual Weak Accuracy**. Some correct process is never suspected.
- **Eventual Weak Accuracy**. There is a time after which some correct process is never suspected by correct processes.

Combined with the completeness property, these accuracy properties define the following two classes of failure detectors [4]:

- $\mathcal{S}$: The class of *Strong* failure detectors. This class contains all the failure detectors that satisfy the strong completeness property and the perpetual weak accuracy property.
- $\diamond\mathcal{S}$: The class of *Eventually Strong* failure detectors. This class contains all the failure detectors that satisfy the strong completeness property and the eventual weak accuracy property.

2. Of course, there are $\mathcal{S}$-based and $\diamond\mathcal{S}$-based consensus protocols that are not particular instances of the proposed generic protocol, e.g., the $\mathcal{S}$-based protocol described in [4] and the $\diamond\mathcal{S}$-based protocol described in [23].

3. The "no message loss" assumption is required to ensure the Termination property of the protocol. The "no creation and no alteration" assumptions are required to ensure its Validity and Agreement properties.

Clearly, a failure detector of the class $\mathcal{S}$ also belongs to the class $\diamond\mathcal{S}$. Moreover, it is important to note that any failure detector that belongs to $\mathcal{S}$ or $\diamond\mathcal{S}$ can make an arbitrary number of mistakes.

2.3 The Consensus Problem

In the consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on some value v, in relation with the set of proposed values. More precisely, the *Consensus* problem is defined by the three following properties [4], [7]:

- **Termination**. Every correct process eventually decides on some value.
- **Validity**. If a process decides v, then v was proposed by some process.
- **Agreement**. No two correct processes decide differently.

The agreement property applies only to correct processes. So, it is possible that a process decides on a distinct value just before crashing. *Uniform Consensus* prevents such a possibility. It has the same Termination and Validity properties plus the following agreement property:

- **Uniform Agreement**. No two processes (correct or not) decide differently.

In the following, we are interested in the *Uniform Consensus* problem.

3 A Versatile Family of Consensus Protocols

As announced in the Introduction the generic protocol is based on two versatility dimensions. The first dimension is related to the message exchange pattern (See Section 3.2), while the second is related to the properties of the underlying failure detector (See Section 3.3).

As in a lot of failure detector-based consensus protocols [4], [12], [17], [18], [23], [25], the processes of the proposed versatile protocol proceed in asynchronous rounds, each round being coordinated by a predetermined process, namely, the round r is coordinated by the process whose identity is $coord(r)$. In order that no two processes associate different coordinators with the same round r the function $coord()$ has to be deterministic.[4]

The round number can be seen as a global logical clock that measures the progress of the computation. The aim of a round is to allow its coordinator to impose a value as the decided value. This will happen if the coordinator is correct and is not suspected by the other processes during the round. To attain this goal, without violating the agreement property when the goal cannot be attained, a round is made up of two phases. During the first phase, the round coordinator sends its estimate of the decision value to all the processes. Then, during the second phase, the processes exchange estimate values according to the message exchange pattern associated with that round. If a process is allowed to test a decision predicate and this predicate is satisfied at the end of a round, the corresponding process decides and terminates. Otherwise, the process proceeds to the next round. The right to test the termination predicate is actually related to the message exchange pattern.

The termination property of the protocol follows from the use of the rotating coordinator paradigm combined with the accuracy property of the underlying failure detector. The main difficulty that the protocol has to solve consists in ensuring that the agreement property is never violated, despite the fact that different processes decide during distinct rounds (due to crashes or erroneous suspicions).

The three sections that follow describe in more details the principles and the behavior of the generic protocol.

3.1 A Two-Phase per Round Protocol

A process p_i starts a consensus execution by invoking Consensus (v_i) (Fig. 1). It terminates it when it executes the statement $return$ which provides it with the decided value (lines 14 and 16).

Each process p_i manages three local variables that define its local state during a round. Those are r_i (current round number), est_i (current estimate of the decision value, initially, equal to v_i, the value proposed by p_i), and ts_i, a timestamp whose value is a round number.

As previously indicated, a round r is made of two phases. During the first phase (lines 4-6), the coordinator of r (the process p_{cc} with $cc = coord(r)$) proposes its current estimate (est_{cc}) to all processes (by broadcasting PROP (r, est_{cc}) at line 4) in order that this estimate becomes the decided value. A process p_i that receives such a PROP (r, v) message adopts it, by updating est_i to v, and ts_i to r_i. So, for a process p_i, the timestamp (ts_i) indicates the last round during which its current estimate of the decision value has been updated (note that this update has been entailed by the reception of a PROP message).

A first phase similar to the previous one is encountered in other failure detector-based consensus protocols [4], [12], [17], [23]. The originality of the approach and the versatility dimension of the proposed protocol appears in its second phase which is the core of the protocol. This phase makes the protocol different from and more general than previous proposals. This dimension lies in the definition of 1) a pair of sets (D, A) which are related to the message exchange pattern and 2) a set (Q_i) with an associated integer ($good(Q_i)$) which are related to the class of the underlying failure detector.

3.2 Versatility Dimension Related to the Message Exchange Pattern

The versatility dimension related to the message exchange pattern is captured by two sets of processes, denoted D and A. These sets are defined for each round separately. In order for the processes to terminate correctly, these sets cannot be arbitrarily defined. According to the round in which they are used, they have to satisfy some constraints.

- The set D is related to the liveness property of the protocol (termination property of the consensus). D stands for *Decision-makers*. It is the set of processes

4. Here, the important point is that the identity of the process that is never suspected (from the beginning if the failure detector belongs to $\mathcal{S}$, or after some time if it belongs to $\diamond\mathcal{S}$) be output by $coord$. A simple way not to miss it is to consider that $coord(r)$ is $(r \bmod n) + 1$ (as in [4], [12], [17], [23]).

```
Function Consensus(v_i)

Task T1:
(1)  r_i ← 0; est_i ← v_i; ts_i ← 0;
(2)  while true do
(3)      r_i ← r_i + 1; let cc =coord(r_i);
         % If the failure detector ∈ S: 1 ≤ r_i ≤ n. If it ∈ ◇S: 1 ≤ r_i < +∞ %

         ———— Phase 1 of round r: from p_cc to all ————————————————————
(4)      if (i = cc) then broadcast PROP(r_i, est_i) endif;
(5)      wait until (PROP(r_i, v) has been received from p_cc ∨ cc ∈ suspected_i);
(6)      if (PROP(r_i, v) received from p_cc) then est_i ← v; ts_i ← r_i endif;

         ———— Phase 2 of round r: from all to D ∪ A ————————————————————
(7)      let set D =decider(r_i);     % D is any set such that {p_cc} ⊆ D ⊆ Π %
(8)      let set A =agr_keeper(r_i); % A is any set such that {p_nc} ⊆ A ⊆ Π where nc =coord(r_i + 1) %
(9)      multicast ECHO(r_i, est_i, ts_i) to (D ∪ A);
(10)     if (p_i∈ D ∪ A) then
         % Q_i -line 11- is the set of processes from which p_i must wait for ECHO messages %
         % good(Q_i) -line 13- is the number of "good" ECHO messages p_i has to receive to decide %
         % An ECHO(r_i, est, ts) message is "good" if r_i = ts %
(11)         wait until (∃Q_i : P(Q_i) ∧ ECHO(r_i, est, ts) has been rec. from each process ∈ Q_i);
(12)         if (i ≠ cc) then est_i ← the est received with highest ts endif;
(13)         if ((p_i ∈ D) ∧ (good(Q_i) ECHO messages from Q_i are such that ts = r_i))
(14)                                  then ∀j ≠ i: send DECISION(est_i) to p_j; return(est_i) endif endif
(15) endwhile

Task T2:
(16) upon receipt of DECISION(est) from p_k: ∀j ≠ i, k: send DECISION(est) to p_j; return(est)
```

Fig. 1. The Versatile Failure Detector-Based Consensus Protocol.

that have to check the decision predicate that allows them to know if they can decide during that round (line 13).[5] For a round r, the set D is defined by the function decider(r) that has to satisfy the following two constraints (for the protocol to terminate):

- decider(r) is deterministic. (Hence, during r, all processes have the same D.)
- decider(r) contains the coordinator of r.

- The set A is related to the safety property of the protocol (uniform agreement property of the consensus). A stands for *Agreement keepers*. As we stressed before, a main issue of consensus protocols is to ensure the agreement property will never be violated despite asynchrony, failures and erroneous suspicions. Actually, it is possible that some processes decide during a round, while other processes decide during another round. If a value has been decided during a round, the role of the processes of A is to ensure no other value be decided during later rounds.

  A is defined by a deterministic function agr_keeper(r). If a value (est_i) is decided by a process p_i (line 12), that value has the highest timestamp (namely, $ts_i = r_i$, the current round number, line 13). So, to maintain the agreement property, any process $p_j \in A$ is required to update its est_j variable to the estimate it has received with the highest timestamp (line 4). Let p_{nc} be the coordinator of the next round, namely, $r + 1$. As, during that round, p_{nc} will try to impose its estimate as the decision value (see the first phase) it must have a "correct" estimate. Hence, at each round, A can be any set of processes including p_{nc}.

So, after it has finished the first phase of the round r, a process p_i sends its current state, namely, the pair (est_i, ts_i) (by multicasting an ECHO(r, est_i, ts_i) message) to the processes of $A \cup D$, i.e., the processes that are the *Decision-makers* for that round and the *Agreement keepers* for the next round (line 9). It follows that the sets A and D (results of the function calls decider(r) and agr_keeper(r)) define the message exchange pattern of the second phase. This pattern can change from round to round (let us note that in the protocols described in [4], [12], [17], [23] the message exchange pattern is defined once for all and the same way for all rounds).

Let us consider a few "extreme" cases. If $(\forall r)$ we define $A = D = \Pi$, we get the "fully distributed" message

5. A process decides "directly" during a round r, if the predicate is true at lines 13 and 14 It decides "indirectly" if it decides upon receiving a decision message at line 16. So, during a round r, only processes of D can decide directly. If D was empty, no process could decide. That is why the definition of D is crucial for the protocol liveness. The "centralized" protocol presented in [4] allows, during each round, only the current round coordinator to decide directly. The "distributed" protocols presented in [12], [17], [23] allow, during each round, every process to decide directly.

exchange pattern for the second phase of all rounds, whose cost is n^2 messages per round. Its advantage is that, at any round, any process can test the decision predicate and accordingly be allowed to decide directly. If ($\forall r$), we define $D = \{p_{cc}\}$ and $A = \{p_{nc}\}$, we get a message exchange pattern that is "centralized" at the current round coordinator; its cost is $2(n-1)$ messages per round. Its advantage is its lower message cost. This shows that, during the second phase, the versatility of the generic protocol offers a tradeoff between the cost of the message exchange pattern of the round and the number of processes that can test the decision predicate and possibly decide directly.

3.3 Versatility Dimension Related to the Class of the Underlying Failure Detector

This versatility dimension (that we started investigating in [17]) consists in the definition of a set and a number for each process that receives messages during the second phase. Let us consider a round r. Each process p_i has a set Q_i and an associated number $good(Q_i)$. Q_i is the quorum set defining the processes from which p_i has to receive ECHO messages (line 11).[6] The actual value of Q_i depends on the state of the system during r (including the current output of the underlying failure detector FD). The values of the quorum sets have to be such that they do not prevent termination of the protocol while ensuring agreement is never violated.

For a process p_i not to block forever while it is waiting for ECHO messages, Q_i has to be be *live*, this means it has to include only noncrashed processes. On the other side, in order to preserve the agreement property, the quorums Q_i and Q_j of two processes have to be *safe*, i.e., they have to include at least one common process. Let Q_i^r be the quorum of p_i at round r and $crashed_i^r$ the processes that have crashed when p_i is at round r. The previous constraints on the quorum definition can be expressed as follows:

- Quorum Safety: $\forall\, r, i, j : Q_i^r \cap Q_j^r \neq \emptyset$.
- Quorum Liveness: $\forall\, r, i : Q_i^r \cap crashed_i^r = \emptyset$.

A way to satisfy these constraints is to dynamically define, during each round, the quorums as follows:

- If FD $\in \mathcal{S}$: Q_i is the set of processes that are not currently suspected by p_i. This ensures the quorum safety as any quorum includes the correct process that is never suspected. The quorum liveness is due to the completeness of FD.
- If FD $\in \diamond\mathcal{S}$ (in that case $f < n/2$ is assumed [4]): Q_i is a set of $(n-f)$ processes. As $n > 2f$, this ensures the quorum safety. Moreover, as there are at least $(n-f)$ correct processes, there is always a live quorum defining Q_i.

As Q_i is dynamically defined and p_i has to wait for messages from the processes in Q_i, we use a predicate $\mathcal{P}$ defined as follows: When FD $\in \mathcal{S}$, $\mathcal{P}(Q_i)$ holds if Q_i contains the processes not currently suspected. When FD $\in \diamond\mathcal{S}$, $\mathcal{P}(Q_i)$ holds if Q_i contains $(n-f)$ processes.

For a process p_i to decide "directly" during a round r, it has to belong to the set of decision-makers for that round (D) and the following decision predicate has to be satisfied (where $good(Q_i)$ is a number):

$$good(Q_i)\text{ECHO messages from processes of } Q_i \text{ are such that } ts = r.$$

This means that enough "good" ECHO messages have been received by p_i. An ECHO message is good if it carries a timestamp (ts) equal to the current round number (r_i). The value of "enough" is directly related to the agreement property and, consequently, (as the definition of Q_i) depends on the class to which belongs the underlying failure detector FD:

- If FD $\in \mathcal{S}$, then $good\ (Q_i) = |Q_i|$ (this ensures that the process that is never suspected knows the decided value).
- If FD $\in \diamond\mathcal{S}$, then $good(Q_i) = (f+1)$ (as $n > 2f$, i.e., $(n-f) \geq (f+1)$, this ensures that at least one correct process knows the decided value).

The proof will show that these values of $good(Q_i)$ allow processes to eventually decide (liveness) without the agreement property being compromised (safety).

3.4 Structure of the Protocol

A process that has decided stops participating in the protocol. Hence, to prevent a process from blocking forever (i.e., waiting for a value from a process that has already decided, see below), a process that decides, uses a reliable broadcast [9] to disseminate its decision value. To this end the Consensus function is made of two tasks, namely, $T1$ and $T2$. $T1$ implements the previous discussion. Line 14 and $T2$ implement the reliable broadcast. Let us note that the task $T2$ is executed at most once per consensus execution.

This paragraph shows why a reliable broadcast of the DECISION messages is necessary. Let us assume that, just before deciding, a process only sends to the other processes a DECISION message carrying the decided value. Let us consider a system equipped with a failure detector FD $\in \diamond\mathcal{S}$, made up of five processes ($p_1, \ldots, p_5$), at most $f = 2$ may crash. Let us consider the case where p_5 has crashed, $p_1, \ldots, p_4$ are executing the round r, decider$(r) = \{p_1\}$, and assume that, during r, p_1 can directly decide. So, p_1 executes line 14, and starts sending DECISION messages. The following scenario can occur: p_1 sends a DECISION message to p_2 and crashes before sending it to p_3 and p_4. In that case, when it receives the DECISION message, p_2 decides and stops participating in the protocol. It follows that p_3 and p_4 constitute a minority of correct processes that is prevented from progressing (and, hence, prevented from deciding). A way to prevent this bad scenario, is to demand that p_2 forwards the DECISION message before deciding. This is systematically realized by implementing a reliable broadcast of the DECISION messages.

4 Proof

4.1 Validity

Theorem 1. *If a process p_i decides v, then v was proposed by some process.*

Proof. Initially, ($r = 0$) all est_i local variables contain proposed values (line 1). The proof is by induction on

6. Let us notice that, for a process p_i and a round r, the quorum set Q_i is relevant only if p_i waits for ECHO messages during r, i.e., if $p_i \in D \cup A$.

the round number. Let us assume that all estimates contain proposed values at the end of $r-1$. We show they contain proposed values at the end of r.

An est_i local variable is updated at line 6 or at line 12. At line 6, est_i is updated to the estimate of the round r coordinator, which is the estimate value this process had at the end of $r-1$. By the induction assumption, this is a proposed value. So, at the end of the first phase of r, all estimate values contain proposed values.

During the second phase of r, if est_i is updated at line 12, it takes the value of an estimate that has been updated during some round, namely, ts. If $ts < r$, this update occurred during the first or the second phase of ts. If $ts = r$, this update occurred during the first phase of r. Hence, at the end of r, each est_i variable contains a value initially proposed by a process.

As a decided value is the value of an est_i variable (lines 14 and 16), the theorem follows. □

4.2 Termination

Lemma 1. *If no process decides during $r' \leq r$, then all correct processes start $r+1$.*

Proof. The proof is by contradiction. Let us assume that no process decided during a round $r' < r$, where r is the smallest round number during which a correct process p_i blocks forever. So, p_i is blocked at line 5 or 11 (in a **wait** statement).

First, let us examine the case where the correct process p_i blocks forever at line 5. If $i = \text{coord}(r) = cc$, p_i cannot block forever at line 5, as it receives the message it broadcast at line 4. If $i \neq cc$, then: 1) either p_i suspects p_{cc}, 2) or p_i never suspects p_{cc}. In case 1, p_i does not block. In case 2, due to the strong completeness property, this means that p_{cc} is correct. From the previous observation, p_{cc} has broadcast its current estimate at line 4 and p_i eventually receives it (reliable channel assumption). It follows that during round r no correct process p_i remains blocked forever at line 5. Consequently, all correct processes send a ECHO message at line 9.

Now, let us consider the case where p_i blocks forever at line 11; so $p_i \in D \cup A$. We consider two cases according to the class of the underlying failure detector. In both cases, the proof uses the fact the functions decider(r) and agr_keeper(r) are deterministic (assumption $\mathcal{A}$): this ensures that the ECHO messages are sent to the processes that wait for them.

- The failure detector belongs to $\mathcal{S}$ and $f \leq n-1$. In that case, Q_i is the set of processes that p_i does not suspect. From $\mathcal{A}$ and the fact that no correct process is blocked forever at line 5, it follows that each correct process sends a ECHO$(r,-,-)$ message to p_i at line 9. Hence, by additionally considering the strong completeness property of the failure detector, we conclude that p_i will receive ECHO$(r,-,-)$ message from all the processes it does not suspect. Consequently, p_i cannot block at line 11.
- The failure detector belongs to the class $\diamond\mathcal{S}$ and $f < n/2$. In that case, Q_i is a first set of $(n-f)$ processes from which p_i receives ECHO$(r,-,-)$ messages. Due to 1) the fact there are at least $(n-f)$ correct processes, 2) those do not block forever at line 5, and 3) assumption $\mathcal{A}$, it follows that at least $(n-f)$ processes sends a ECHO$(r,-,-)$ message to p_i. Consequently, p_i cannot block at line 11.

As no correct process blocks forever during round r, they all proceed to the round $r+1$, which contradicts the initial assumption. □

Theorem 2. *If a process p_i is correct, then it decides.*

Proof. If a (correct or not) process decides, then due to the lines 14 and 16 (that implement a reliable broadcast) all correct processes decide.

So, suppose that no process decides. The proof is by contradiction. Due to the accuracy property of the underlying failure detector, there is a time t after which there is a correct process that is never suspected. (Note that $t = 0$ if the failure detector belongs to $\mathcal{S}$ and $t \geq 0$ if it belongs to $\diamond\mathcal{S}$, assuming the protocol starts executing at time $t = 0$.) Let p_x be this process. Moreover, let r be the first round that starts after t and that is coordinated by p_x. As by assumption no process decides, due to Lemma 1, all the correct processes eventually start round r.

The process p_x starts round r by broadcasting PROP(r, est_x) and, as, after t, p_x is not suspected, any process that has not crashed echoes the PROP(r, est_x) message by sending a ECHO(r, est_x, r) message to processes of $D \cup A$, hence to p_x (p_x being the coordinator of r, due to the definition of D, it belongs to D).[7] As seen in the proof of Lemma 1, p_x does not block at line 11 and due to the previous observation, all the ECHO messages it receives carry the same triple (r, est_x, r). As in all those triples, the round number (first field) is equal to the timestamp (third field), we have $good(Q_x) = |Q_x|$ if the failure detector belongs to $\mathcal{S}$ and $good(Q_x) = (n-f) \geq (f+1)$ if it belongs to $\diamond\mathcal{S}$. Consequently, when p_x checks the decision predicate, it gets the value $true$ and decides, which contradicts the assumption that no process decides. □

4.3 Uniform Agreement

Lemma 2. *If the failure detector belongs to $\mathcal{S}$, let $r \geq 1$ be the first round during which a correct process p_x that is never suspected, sends ECHO(r, v, r).*[8]

If the failure detector belongs to $\diamond\mathcal{S}$, let $r \geq 1$ be the first round during which $(f+1)$ processes sends ECHO(r, v, r).

We have:

1. *No process decides before r.*
2. *Let r' be any round $\geq r$. If the coordinator of r' sends a PROP message, then this message carries v (i.e., the message is PROP(r', v)).*

7. Actually, the rest of the prof remains correct if, instead of considering p_x, we now consider any correct process that belongs to D.

8. This means that r is the first round during which p_x receives a value v carried by a PROP message and, consequently, echoes it by sending ECHO messages. Using the traditional terminology, from that round v is "locked" and, consequently, will be the decided value.

Proof. Proof of 1. To decide at line 13 during a round r, a process has to receive ECHO messages carrying a timestamp equal to r from some set of processes.

- If the failure detector belongs to $\mathcal{S}$, the set has to includes p_x.
- If the failure detector belongs to $\diamond\mathcal{S}$, this set has to include at least $(f+1)$ processes.

The impossibility for a process to decide before r comes from the combination of the previous observation and the definition of r ("first round during which ...").

The proof of 2 is by induction on the round number.

- Base case: $r' = r$. Due to the first phase of the protocol, a process sends ECHO(r, v, r) during a round r, only if it previously received a PROP(r, v) message. Hence, the lemma trivially holds.
- Induction case: $r' > r$. Let us assume the lemma holds for any round k such that $r \leq k \leq r'$. We show it is true for $r'+1$.
  Due to the induction assumption, all PROP(k, w) messages with $r \leq k \leq r'$ are such that $w = v$. Let us consider two sets of processes:
  - The set R. This set includes all processes p_j that have received a PROP(k, w) message with $r \leq k \leq r'$. $\forall p_j \in R$: p_j updated est_j to $w = v$ and ts_j to a round number k' such that $r \leq k' \leq r'$.
  - The set T. This set includes all the processes p_j that have not received a PROP(k, w) message with $r \leq k \leq r'$. Consequently, $\forall p_j \in T$: ts_j has a value $< r$.

Now, let us consider the behavior of a process $p_y \in A$ during the second phase of round r'. If $p_y = p_{cc}$ (where $cc = \text{coord}(r')$) it has $est_y = v$ as, due the induction assumption, it sent PROP(r', v). Now, let us examine the case $p_y \neq p_{cc}$. It first receives ECHO messages from all the processes of Q_y (line 11). We claim that, during r', $Q_y \cap R \neq \emptyset$.

- Case 1. The failure detector belongs to $\mathcal{S}$. In that case, Q_y includes the correct processes that are never suspected, hence, $p_x \in Q_y$. Due to the definition of "r", p_x belongs to R (p_x sent an ECHO(r, v, r) message during r). Hence, $Q_y \cap R \neq \emptyset$.
- Case 2. The failure detector belongs to $\diamond\mathcal{S}$. In that case, due to the definition of the round "r", $(f+1)$ processes sent ECHO(r, v, r) during that round. Let p_j be any of those $(f+1)$ processes. Before sending ECHO(r, v, r), p_j executed the first phase of r and, consequently, received a PROP(r, v) message (and accordingly updated its timestamp ts_j). Hence, $p_j \in R$. As p_j is any process of a set of $(f+1)$ processes, we conclude that $|R| \geq (f+1)$. As $|Q_y| = (n-f)$ and $|R| \geq (f+1)$, we have $Q_y \cap R \neq \emptyset$.

Due to $Q_y \cap R \neq \emptyset$, the highest timestamp contained in the ECHO messages received from the processes of Q_y cannot come from a process $\in T$: it necessarily comes from a process of R. Hence, when at line 12, p_y updates est_y to the est received with the highest timestamp, it sets it to v. Finally, due to the definition of agr_keeper(r'), the coordinator of $r'+1$ belongs to A. It follows that if, during $r'+1$, its coordinator broadcasts a PROP message, this message carries the pair $(r'+1, v)$. □

Theorem 3. *No two processes decide differently.*

Proof. If a process decides a value at line 16, then this value has been decided by another process at line 16. So, we only consider values decided at line 14.

Let p_i and p_j be two processes that decide $v1$ and $v2$ during the rounds $r1$ and $r2$, respectively. As p_i (respectively, p_j) decides during $r1$ (respectively, $r2$), it received ECHO$(r1, v1, r1)$ (respectively, ECHO$(r2, v2, r2)$) during $r1$ (respectively, $r2$). If follows from the first phase of the protocol that, if a process receives an ECHO$(r1, v1, r1)$ (respectively, ECHO$(r2, v2, r2)$) message, then the coordinator of $r1$ (respectively, $r2$) sent a PROP$(r1, v1)$ (respectively, PROP$(r2, v2)$) message. Without loss of generality let us assume $r1 \leq r2$.

Let r be the round characterized in Lemma 2. We conclude from this Lemma that the messages PROP(r, v), PROP$(r1, v1)$ and PROP$(r2, v2)$ are such that $r \leq r1 \leq r2$ and $v = v1 = v2$. □

5 Instantiating the Protocol

5.1 Deriving Particular Protocols

As indicated in Sections 3.2 and 3.3, the versatility of the protocol allows it to be instantiated by providing independently 1) a failure detector FD and 2) a definition for the functions decider and agr_keeper. Here, we sum up and briefly discuss some possible instantiations. In the following: the sets D, A, Q_i and the number $good(Q_i)$ are related to a round r.

Class of the failure detector. The instantiation of the protocol with a particular unreliable failure detector has already been addressed in Section 3.3. This paragraph summarizes the previous discussion.

- FD $\in \mathcal{S}$: then, Q_i is the set of the processes that are not suspected by p_i when it is waiting for ECHO messages (so, it receives ECHO messages from them); $good(Q_i)$ is equal to $|Q_i|$. In that case, we must have $f \leq n-1$.
- FD $\in \diamond\mathcal{S}$: then, Q_i is a set made up of $(n-f)$ processes and $good(Q_i) = (f+1)$. In that case, we must have $f < n/2$ [4].

As the class of perfect (respectively, eventually perfect) failure detectors is a subclass of $\mathcal{S}$ (respectively, $\diamond\mathcal{S}$), they can also be used when instantiating the protocol.

Tuning the message exchange pattern. This pattern is defined on a round basis by the decider and agr_keeper functions (Section 3.2). As we have seen, decider(r) (respectively, agr_keeper(r)) has to include p_{cc}, the coordinator of the round r (respectively, p_{nc}, the coordinator of the round $r+1$). We examine a few cases.

- Case 1. decider$(r) = \{p_{cc}\}$ and agr_keeper$(r) = \{p_{nc}\}$. This case provides a protocol where the message exchange pattern of the round r is centralized. In that case, the message cost of a round is $3(n-1)$ messages

(namely, $(n-1)$ PROP messages from the coordinator, plus $2(n-1)$ ECHO messages to the current and the next coordinators).

The instantiation of the protocol with FD $\in \mathcal{S}$ and the previous definitions for decider and agr_keeper gives rise to a new $\mathcal{S}$-based consensus protocol whose cost is $3(n-1)$ messages per round. This protocol requires at most n rounds (each involving one or two communication steps) and allows for early decision when there are no erroneous suspicions.

The instantiation of the protocol with the previous function definitions and FD $\in \diamond\mathcal{S}$, provides a protocol whose behavior is the same as Chandra-Toueg's $\diamond\mathcal{S}$-based consensus protocol (see Section 5.2).

- Case 2. decider$(r) =$ agr_keeper$(r) = \Pi$. In that case, the message exchange pattern of the round r is fully distributed and a round costs n^2 messages (the ECHO messages from the coordinator can be saved as they carry no new information with respect to the PROP message it previously sent).

  If we consider decider$(r) =$ agr_keeper$(r) = \Pi$, $\forall r$, we get $\mathcal{S}; / \diamond\mathcal{S}$-based consensus protocols close to the ones introduced in [17].
- Case 3. $\{p_{cc}\} \subset$ decider$(r) \subset \Pi$ and $\{p_{nc}\} \subset$ agr_keeper$(r) \subset \Pi$. In that case, the message exchange pattern of the round r is partially distributed. As announced in the Introduction, this means that it is possible to define a parameter k_r $(1 < k_r \leq n)$ associated with each round r, such that the message cost of the round r is $O(k_r n)$. The instantiations with these definitions of decider and agr_keeper provide new consensus protocols (whichever class ($\mathcal{S}$ or $\diamond\mathcal{S}$) the failure detector belongs to).

On the definition of the functions decider and agr_keeper. This paragraph examines different choices when defining decider and agr_keeper.

- An interesting choice is to have agr_keeper$(r) \subseteq$ decider(r). This allows, during r, any process p_j that belongs to $A =$ agr_keeper(r) to test the decision predicate without additional cost (messages, synchronization, etc.).
- An interesting possibility for the function decider is to combine the two "extreme" cases, in the following way. Let us consider an integer $\alpha \geq 1$ (if the failure detector is tuned to make very few mistakes we can take $\alpha = 1$).

  - If $1 \leq r \leq \alpha$: decider$(r) = \Pi$.
  - If $r > \alpha$: decider$(r) = \{p_{cc}\}$ (the coordinator of r).

  This definition of decider is interesting because, when no crash occurs (the usual case, in practice), the use of a distributed message exchange pattern expedites the decision (all processes are allowed to decide directly). Differently, when there are crashes or due to too many erroneous suspicions, this strategy allows to reduce the number of messages without notably increasing the round time complexity.
- Similarly, simple instantiations for the function agr_keeperconsists of taking the "extreme" values, i.e., $\forall r$: agr_keeper $(r) = \{p_{nc}\}$ or Π.
- The function decider could also be defined according to the current system load, the size of D being increased (respectively, decreased) under light (respectively, heavy) load.

5.2 Revisiting Chandra and Toueg's $\diamond\mathcal{S}$-Based Consensus Protocol

This section shows that the Chandra and Toueg's well-known $\diamond\mathcal{S}$-based consensus protocol (CT) [4] is a particular instance of the versatile protocol. In CT, each round r_i is coordinated by a predetermined process p_{cc} (with $cc \leftarrow (r_i \bmod n) + 1$). During a round the message pattern is centralized (to/from the current round coordinator). More precisely, the messages are exchanged as follows:

1. First, the processes send their estimates of the decision value to the round coordinator (ESTIMATE messages, phase 1).
2. The coordinator then selects the estimate with the largest timestamp and sends it to the processes (phase 2).
3. Each process waits for the ESTIMATE message from the coordinator. If it receives it, it sends back an ACK message to the round coordinator. If while waiting, it suspects the coordinator, it sends it a NACK message (phase 3). In both cases, the process then proceeds to the next round.
4. If the coordinator receives ACK messages from a majority of processes, it reliably broadcasts a DECISION message carrying the decided value.

This message exchange pattern is represented in Fig. 2a. Let us note that, in the particular case of the first round, the first step of the round can be suppressed (in that case, as there is no previous round, the agreement property cannot be violated and, consequently, the first round coordinator may simply try to impose its estimate value, whatever the other estimates are).

Interestingly, the versatile protocol described in Fig. 1 can be appropriately instantiated to obtain a variant of CT described in Fig. 3 (let us notice that this variant and CT do have the same behavior, they differ only in the way they are written). The functions decider and agr_keeper are defined as follows: For any round r, the output of decider(r) is the current round coordinator, while the output of agr_keeper(r) is systematically the coordinator of the next round. The function coord() is instantiated as $cc = \text{coord}(r_i) = (r_i \bmod n) + 1$. Moreover, nc denotes the next round coordinator. Hence, we have decider$(r) = \{p_{cc}\}$ and agr_keeper$(r) = \{p_{nc}\}$.

The message exchange pattern of the variant is described in Fig. 2b. It is important to see that Figs. 2a and 2b describe the same message exchanges. Let (r, x, y) denote the phase x of round r described in Fig. 2.y (where $y =$ a/b). Phase 2 (respectively, 3) is coordinated by p_2 (respectively, p_3). The phases $(2, 2,$a) and $(2, 1,$b) define the same message exchange pattern (during which the coordinator p_2 broadcasts its estimate). Moreover, The consecutive phases $(2, 3,$a) (during which the processes send ACK/NACK messages to the coordinator) and $(3, 1,$a) (during which the next coordinator p_3 broadcasts its estimate) produce the same message exchange pattern as the phase $(2, 2,$b) (during

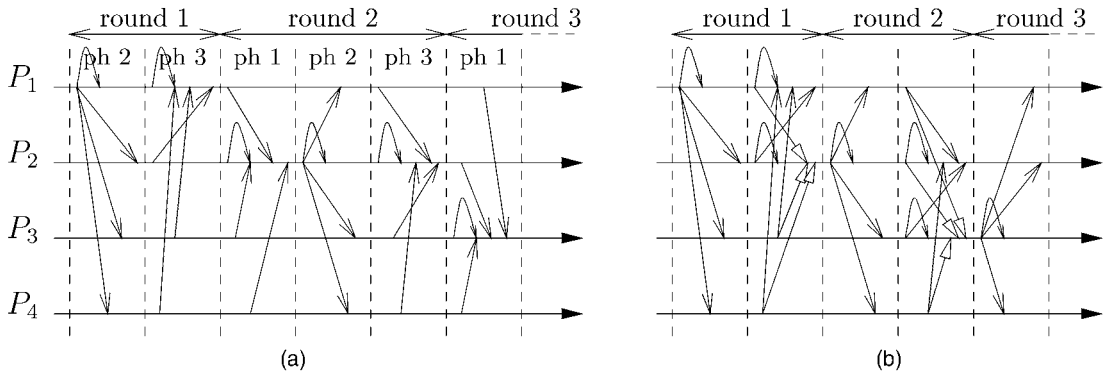


Fig. 2. Rounds in Chandra-Toueg's $\diamond\mathcal{S}$-based Consensus Protocol.

which the current and next coordinators broadcast ECHO messages). More generally, the last exchange of a round r and the first exchange of the round $r+1$ in Fig. 2a are merged in a single message exchange in Fig. 2b, namely, the second message exchange of round r. By unfolding the last phase of each round of Fig. 2b, we get Fig. 2a. As far as the message types are concerned, we have the following: The PROP message sent by the round coordinator (line 4, Fig. 3) is the ESTIMATE message sent in CT. An ECHO message with $ts_i = r_i$ represents a CT ACKmessage, while an ECHO message with $ts_i \neq r_i$ represents a CT NACK message. Finally, the CT ESTIMATE messages sent by the processes to the coordinator are represented here by ECHO messages sent at line 12.

This shows that CT is a member of a large and well-identified family of $\diamond\mathcal{S}$-based consensus protocols. This protocol (with a centralized message exchange pattern) and the protocol with a fully distributed message exchange pattern are actually nothing but the two extreme cases of a very same general protocol.

6 Discussion

6.1 Message Cost

As we have seen, each round of the protocol requires $O(k_r n)$ messages ($1 < k_r \leq n$). In addition to these messages, the protocol uses a reliable broadcast (task $T2$) to prevent a possible deadlock by disseminating DECISION messages. As implemented, the reliable broadcast requires $O(n^2)$ messages. According to the network topology more efficient implementations of reliable broadcast can be designed [21].

When the general protocol is instantiated with a failure detector of the class $\diamond\mathcal{S}$ and $D = \Pi$ at each round, the reliable broadcast of DECISION messages at line 14 (Fig. 1) can actually be suppressed in some circumstances, thereby reducing the message cost of the protocol. More precisely, let us consider the lines 11-14 of Fig. 1 in the context previously defined:

wait until(an ECHO(r_i, est, ts) message has been received from $(n-f)$ processes);
if $(i \neq cc)$ **then** $est_i \leftarrow$ the est received with highest ts **endif**;
if ($(f+1)$ ECHO messages are such that $ts = r_i$)
then $\forall j \neq i$: *send* DECISION(est_i) *to* p_j;
$return(est_i)$ **endif**

We show that the addition of a new line (denoted *) into this sequence of statements allows executions to save reliable broadcast invocations.

wait until (an ECHO(r_i, est, ts) message has been received from $(n-f)$ processes);
if$(i \neq cc)$ **then** $est_i \leftarrow$ the est received with highest ts **endif**;
* **if** ($(2f+1)$ ECHO messages are such that $ts = r_i$)
then $return(est_i)$ **endif**;
if ($(f+1)$ = ECHO messages are such that $ts = r_i$)
then $\forall j \neq i$: *send* DECISION(est_i)
to p_j; $return(est_i)$ **endif**

When, among the $(n-f)$ ECHO(r_i, est, ts) messages received by a process p_i, $(2f+1)$ are such that $ts = r_i$, we can conclude that any process p_j that receives $(n-f)$ ECHO(r_i, est, ts) messages, necessarily receives at least $(f+1)$ ECHO(r_i, est, ts) messages with $ts = r_i$ and, consequently, decides by executing $return$ (at the line (*) or the next line). For this scenario to occur, it is necessary that $n - f \geq 2f + 1$, i.e., $f < n/3$. Interestingly, if each process receives $(2f+1)$ ECHO messages such that $ts = r_i$, then no process broadcasts DECISION messages.

6.2 Reducing the Size of ECHO Messages

The variables r_i and ts_i have a bounded domain when the protocol is instantiated with a failure detector of $\mathcal{S}$. This is due to the fact that, in that case, the number of rounds is bounded. This is no longer true when the failure detector belongs to $\diamond\mathcal{S}$; in that case, the number of rounds is unbounded. We show here that, although the round number (r_i) carried by the ECHO messages cannot be bounded, the timestamp value (ts_i) carried by these messages has a bounded implementation.

We suppose here that a process never suspects itself. Let $d(i)$ be the maximal number of rounds between two successive calls to the function coord that output the same process identity i. Moreover, let $d = max_{1 \leq i \leq n}(d_i)$ (If coord$(r) = ((r-1) \bmod n) + 1$, then $d = n - 1$).

In addition to its control local variables (r_i and ts_i), we supplement p_i with a new local variable δ_i, whose scope is the current round. Its aim is to allow ECHO messages to no

```
Function Consensus(v_i)

Task T1:
(1)  r_i ← 0; est_i ← v_i; ts_i ← 0;
(2)  while true do
(3)      cc ← (r_i mod n) + 1; nc ← ((r_i + 1) mod n) + 1; r_i ← r_i + 1;      % 1 ≤ r_i < +∞ %

         % The current round coordinator (p_cc) proposes its estimate to all %
(4)      if (i = cc) then broadcast PROP(r_i, est_i) endif;
(5)      wait until (PROP(r_i, v) has been received from p_cc ∨ cc ∈ suspected_i);
(6)      if (PROP(r_i, v) received from p_cc) then est_i ← v; ts_i ← r_i endif;

         % Any process positively/negatively answers to the round coordinator (p_cc) %
         % Here the set of decider processes is D = {p_cc} %
(7)      send ECHO(r_i, est_i, ts_i) to p_cc; % (ts_i = r_i) ≡ACK and (ts_i ≠ r_i) ≡ NACK %
         % The current coordinator tries to decide %
(8)      if (i = cc) then wait until ((n − f) ECHO(r_i, est, ts) messages have been received);
(9)                      if (at least f + 1 messages are such that ts = r_i)
(10)                         then ∀j ≠ i: send DECISION(est_i) to p_j; return(est_i) endif
(11)     endif;

         % The coordinator of r_i + 1 has to start that round with a correct estimate. This %
         % is the role of the first step of r_i + 1 in CT. An equivalent behavior is obtained %
         % by shifting this step to be the last step of round r_i, and defining A (the set of %
         % agreement keeper processes) equal to {p_nc} %
(12)     send ECHO(r_i, est_i, ts_i) to p_nc;
(13)     if (i = nc) then wait until ((n − f) ECHO(r_i, est, ts) messages have been received);
(14)                      est_i ← the est received with the highest ts
(15)     endif;
(16) endwhile

Task T2:
(17) upon receipt of DECISION(est) from p_k: ∀j ≠ i, k: send DECISION(est) to p_j; return(est)

The send of lines 7 and 12 are merged in a single multicast in the versatile protocol
```

Fig. 3. Chandra and Toueg's $\diamond\mathcal{S}$-based Consensus Protocol.

longer carry the values of ts_i variables, they instead carry a "differential" value, represented by δ_i. More precisely, δ_i is managed in the following way during a round r.

- Just before line 9, δ_i is set to the differential value $r_i - ts_i$. Note that $\delta_i \geq 0$.
- The multicast of line 9 now sends a triple where δ_i replaces ts_i, namely, ECHO(r_i, est_i, δ_i).
- At line 11, p_i now receives ECHO(r_i, est, δ) and:
  - At line 12: it considers the est associated with the smallest δ.
  - At line 13: it considers the ECHO(r_i, est, δ) messages such that $\delta = 0$.

The reader can easily check that the introduction of these δ_i does not modify the behavior of the protocol.

Theorem 4. *Let us consider a process p_i executing the round r (so, $r_i = r$). We have $r_i - d \leq ts_i \leq r_i$.*

Proof. If p_i coordinates $r_i = r$, then it sets ts_i to r (line 6). If p_i is not the coordinator of r, the theorem follows from 1) the fact that p_i coordinated at least one round (r') such that $r - d \leq r' \leq r - 1$ and 2) as p_i does not suspect itself, it updated ts_i to r' when it coordinated this round (line 6).□

We conclude from this theorem that the domain of any local variable δ_i is bounded, namely, $0 \leq \delta_i \leq d$. So, the timestamp values carried by messages are also bounded. This shows that the CT protocol can use messages that carry a single unbounded value (the round number).

Let us observe that the message exchange pattern of a round r is precisely defined from the values of cc, nc, D, and A (those are deterministically determined from r). Said in another way, this means that each process p_i can determine this pattern from the round number r and accordingly deduce how many messages each process p_j should send to it during r (if p_j has not crashed).

Now, let us consider the case where the channels are FIFO. It follows, from the FIFO property and the fact that a process can compute the message exchange pattern of each round, that any process p_i can compute, for each p_j, how many messages p_j should have sent to it from the beginning until r. As channels are reliable and FIFO, we conclude that p_i can compute the round number of each message it receives from p_j. Consequently, the round number can be

suppressed from messages when channels are FIFO and the messages can then carry only bounded values.

6.3 Boolean Flags versus Timestamps

Now, let us consider the instance of the versatile protocol where $\forall r$: $\mathsf{agr\_keeper}(r) \cup \mathsf{decider}(r) = \Pi$. This means that, during each round, every process p_i updates its est_i variable to the estimate it has received with the highest timestamp (line 12).

Now, let us revisit the round r of the protocol by modifying line 6 and replacing ts_i with a Boolean flag ($flag_i$) indicating whether p_i has received the estimate of the current round coordinator:

(6′) **if** (PROP(r, v) received from p_{cc}) **then** $est_i \leftarrow v$;
$flag_i \leftarrow true$ **else** $flag_i \leftarrow false$ **endif**

Let us consider the second phase. As now $\forall i$: $p_i \in D \cup A = \Pi$, the test at line 10 disappears and any p_i receives ECHO messages, each carrying a triple $(r, est, flag)$. An ECHO(r, est, r') message of the original protocol becomes in the revisited version ECHO$(r, est, true)$ if $r = r'$ and ECHO$(r, est, false)$ if $r' < r$. The lines 12-13 consequently become:

(12′) **if** $(i \neq cc)$ **then** $est_i \leftarrow$ an est received with $flag = true$ **endif**;
(13′) **if** $((p_i \in D) \wedge (good(Q_i)$ ECHO messages from Q_i have $flag = true))$ **then**...

The revisited protocol remains correct. Actually, the replacement of ts_i with $flag_i$ invalidates only the proof of the item 2 of Lemma 2. Its statement remains correct but has to be proved again in this new setting. Here is a proof for the item 2 when Boolean flags replace timestamps.

Proof. Let r be the first round during which a process p_i decides. The decided value v is necessarily the value proposed by the coordinator of r. We show that, for any round $r' \geq r$, the PROP message sent by the coordinator of r' contains the pair (r', v). There are two cases according to the fact the failure detector FD belongs to $\mathcal{S}$ or $\diamond\mathcal{S}$.

- FD $\in \mathcal{S}$. Let p_x be the correct process that is never suspected. As p_i decides during r, it has received ECHO$(r, v, true)$ from all the processes in Q_i, hence from p_x. As no process suspects p_x, any p_j receives this message from p_x during r and updates est_j to v at line 12′. Consequently, no estimate has a value different from v at the end of the round r.
- FD $\in \diamond\mathcal{S}$. In that case, p_i received $(f + 1)$ ECHO$(r, v, true)$ messages. As any process p_j waits for $(n - f)$ ECHO messages, at least one of them carries $(r, v, true)$. So, p_j updates est_j to v at line 12′. Consequently, no estimate has a value different from v at the end of the round r. □

This shows that, when $\forall r$: $(\mathsf{agr\_keeper}(r) \cup \mathsf{decider}(r)) = \Pi$, the timestamps can be replaced by a Boolean flag. This provides an explanation of the fact that the $\diamond\mathcal{S}$-based protocol of [4] needs timestamps, while Boolean flags are sufficient in the $\diamond\mathcal{S}$-based protocols presented in [12], [17], [23].

The intuition that underlies this "Boolean/timestamp" issue lies in the fact that the scope of a Boolean variable $flag_i$ is limited to a single round (during each round, $flag_i$ is first set at line 6′ and then sent with ECHO messages to be used by other processes in the second phase). So, it can carry control information related only to the first phase of the current round. Contrary to a timestamp, it cannot convey, from a round to another, control information whose scope potentially concerns previous rounds (i.e., the past history of the execution).

It is also worth noticing that, from an operational point of view, the condition $\forall r$: $(\mathsf{agr\_keeper}(r) \cup \mathsf{decider}(r)) = \Pi$ defines a global synchronization barrier the processes have to pass over in order to proceed from one round to the next one. Actually, as far as the control information is concerned, this global synchronization allows processes to forget the past (except the round number) when they proceed from a round to the next one.

6.4 Partially Synchronous Systems

Partially synchronous computation models have been introduced in [6] where two models of partial synchrony are considered. One model considers that there are bounds on relative process speeds and message transmission times, but these bounds are unknown. Another partially synchronous model considers these bounds are known but hold only after some unknown time. A weaker partially synchronous model is when there are bounds but they are not known and they hold only after some unknown time. Let PS such a partially synchronous system. It has been shown that the consensus problem can be solved in such systems [6]. An eventually strong failure detector can also be easily built in a PS system [4].

We show here that the consensus problem can be solved in a PS system with an appropriate instantiation of the generic protocol presented in Section 3. Interestingly, this instantiation is "direct" in the sense that it does not explicitly build a failure detector.

As it has been noticed when presenting the protocol, the round number can be seen as a logical global clock that measures the protocol progress. This clock can be used in the following way to replace the set $suspected_i$ by a local timer ($timer_i$). Let $f(r)$ be an integer function whose relevant property is to increase when r increases (e.g., $f(r) = r$). Line 5 of Fig. 1:

wait until (PROP(r_i, v) has been received from $p_{cc} \vee cc \in suspected_i$)

is replaced by:

set $timer_i$ **to** $f(r_i)$;
wait until (PROP(r_i, v) has been received from $p_{cc} \vee timer_i \leq 0$)

Due to the "eventual" attribute of the underlying PS system, this instantiation requires $f < n/2$ (as the instantiation with $\diamond\mathcal{S}$). The proof that this modified protocol works in a PS system is close to the one that has been done when the underlying failure detector belongs to $\diamond\mathcal{S}$. Basically, due to the existence of eventual bounds, there is eventually a round during which the PROP messages from a correct coordinator are received and taken into account by all the noncrashed processes.

Remark. The protocol could also be modified to work in the crash/recovery model. In that case, unreliable failure detectors suited to this model have to be defined [1], [11], [20]. The modification of the generic protocol would

follow the same lines as the ones we followed in [11] to adapt the $\diamond\mathcal{S}$-based consensus protocol described in [12] to the crash/recovery model.

7 Conclusion

This paper addressed the design of consensus protocols in asynchronous distributed systems equipped with unreliable failure detectors. It presented a unifying approach based on two orthogonal versatility dimensions. The first lies in the class of the underlying failure detector. An instantiation can consider any failure detector of $\mathcal{S}$ (provided $f \leq n-1$) or $\diamond\mathcal{S}$ (provided $f < n/2$). The second versatility dimension concerns the message exchange pattern the processes follow during a round. This pattern and, consequently, the round message cost, can be defined for each round separately. It can vary from $O(n)$ (in that case the message exchange pattern is centralized) to $O(n^2)$ (in that case the message exchange pattern is fully distributed). The resulting versatile protocol gives rise to a family of failure detector-based consensus protocols. This large and well-identified family includes at once new protocols and some well-known existing protocols (e.g., [4]).

The proposed versatility-oriented approach is also interesting from a methodological point of view. It provides a very precise characterization of the two sets of processes that, during a round, have to receive messages for a decision to be taken (liveness) and for a single value to be decided (safety), respectively.

To conclude, let us notice the two following points: First, it is possible to combine the failure detector approach and the randomization approach to solve consensus [2], [19]. Second, a new *condition-based* approach to solve consensus has recently been proposed [15]. It consists in identifying the sets of input vectors for which the consensus problem is solvable despite up to f process crashes. Interestingly, these sets of conditions define a hierarchy [16].

Appendix A

Principles of $\mathcal{S}/\diamond\mathcal{S}$-Based Consensus Protocols

As noticed in the Introduction, the $\mathcal{S}/\diamond\mathcal{S}$-based consensus protocols designed in [4], [8], [12], [18], [23], [25] share some design principles (some of which have been used in the generic protocol). Let FD be the underlying failure detector used by the considered protocol. Those principles are:

- (RC) Round coordinator (FD $\in \mathcal{S}$ or $\diamond\mathcal{S}$). To benefit from the accuracy property of the underlying failure detector, these protocols are based on the rotating coordinator paradigm.
  Each round is managed by a process (the round coordinator) that tries to impose its estimate as the decision value. If each process is repeatedly chosen to coordinate a round, the accuracy property guarantees that a process will succeed in imposing its estimate as the decision value.
- (C) No deadlock before a decision is taken (FD $\in \mathcal{S}$ or $\diamond\mathcal{S}$). The completeness property of the failure detector is used to prevent a process from blocking forever in a round before a value is decided.
  More precisely, the completeness property allows, during each round, a process to receive a value from the current coordinator or suspect it.

TABLE 1
Protocols and Their Principles

Protocols	FD	Principles
[18, 25]	$\mathcal{S}$	RC + C + A_p + FT
[8]	$\mathcal{S}$	RC + C + A_p + ET
[4, 12, 23]	$\diamond\mathcal{S}$	RC + C + A_e + ET

- Agreement. This property is ensured differently according to the underlying failure detector:
  - (A_p) FP $\in \mathcal{S}$: In that case, the agreement property comes from the weak accuracy property. A process that is not suspected imposes its estimate as the decided value.
  - (A_e) FP $\in \diamond\mathcal{S}$: In that case, the agreement is a consequence of the assumption $f < n/2$ (which is necessary with "eventual" failure detectors). The protocols use the fact that two majorities always intersect, to guarantee that no two processes decide differently.
- Termination. There are two cases:
  - (ET) Early termination. In that case, the protocol does not force the processes to execute a fixed number of rounds: a process terminates as soon some local predicate becomes true.
    It is the accuracy property of FD that ensures that this predicate eventually becomes true. Basically, as soon as a correct coordinator is not suspected, its value is imposed to all processes.
    It is important to note that early termination has the following consequence. It is possible that not all the processes decide during the same round, some processes deciding during a round r while others proceed to $r+1$. As a process that decides stops participating in the protocol, it is possible that processes executing $r+1$ wait forever message from it (e.g., this can occurs if this process is correct, is the coordinator of $r+1$ and is not suspected). This possibility of deadlock is prevented by forcing a deciding process to reliably broadcast its decision value.
  - (FT) Termination in a fixed number of rounds. First, let us note that this is possible only when the accuracy property of FD is not eventual. In that case, the processes have to execute n rounds in order not to miss the process that is not suspected. As all the processes execute the same number of rounds, no deadlock situation similar to the previous one can occur.

Table 1 indicates the principles used in each protocol. (Let us remark that the $\mathcal{S}$-based consensus protocol described in [4] is not explicitly based on the round coordinator paradigm.)

Acknowledgments

The authors are grateful to the referees whose comments were instrumental in improving the paper.

REFERENCES

[1] M.K. Aguilera, W. Chen, and S. Toueg, "Failure Detection and Consensus in the Crash-Recovery Model," *Distributed Computing,* vol. 13, no. 2, pp. 99-125, 2000.

[2] M.K. Aguilera and S. Toueg, "Failure Detection and Randomization: a Hybrid Approach to Solve Consensus," *SIAM J. Computing,* vol. 28, no. 3, pp. 890-903, 1998.

[3] J. Brzezinsky, J.-M. Hélary, M. Raynal, and M. Singhal, "Deadlock Models and a General Algorithm for Distributed Deadlock Detection," *J. Parallel and Distributed Computing,* vol. 31, no. 2, pp. 112-125, 1995.

[4] T. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM,* vol. 43, no. 2, pp. 225-267, Mar. 1996.

[5] T. Chandra, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," *J. ACM,* vol. 43, no. 4, pp. 685-722, July 1996.

[6] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM,* vol. 35, no. 2, pp. 288-323, 1988.

[7] M.J. Fischer, N. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM,* vol. 32, no. 2, pp. 374-382, Apr. 1985.

[8] F. Greve, M. Hurfin, R. Macêdo, and M. Raynal, "Time and Message Efficient S-Based Consensus," *Brief announcement, Proc. 19th ACM SIGACT-SIGOPS Int'l Symp. Principles of Distributed Computing (PODC '00),* p. 332, July 2000.

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

[10] J.-M. Hélary, A. Mostéfaoui, and M. Raynal, "A General Scheme for Token- and Tree-Based Distributed Mutual Exclusion Algorithms," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 11, pp. 1185-1196, 1994.

[11] M. Hurfin, A. Mostéfaoui, and M. Raynal, "Consensus in Asynchronous Systems Where Processes Can Crash and Recover," *Proc. 17th IEEE Symp. Reliable Distributed Systems,* pp. 280-286, Oct. 1998.

[12] M. Hurfin and M. Raynal, "A Simple and Fast Asynchronous Consensus Protocol Based on a Weak Failure Detector," *Distributed Computing,* vol. 12, no. 4, pp. 209-223, 1999.

[13] A.D. Kshemkalyani and M. Singhal, "On the Characterization and Correctness of Distributed Deadlock Detection," *J. Parallel and Distributed Computing,* vol. 22, no. 1, pp. 44-59, 1994.

[14] N. Lynch, *Distributed Algorithms.* San Francisco: Morgan Kaufmann pp. 1-872, 1996.

[15] A. Mostéfaoui, S. Rajsbaum, and M. Raynal, "Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems," *Proc. 33rd ACM Symp. Theory of Computing (STOC '01),* pp. 153-162, July 2001.

[16] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and M. Roy, "A Hierarchy of Conditions for Consensus Solvability," *Proc. 20th ACM Symp. Principles of Distributed Computing (PODC '01),* pp. 151-160, Aug. 2001.

[17] A. Mostéfaoui and M. Raynal, "Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach," *Proc. 13th Int'l Symp. Distributed Computing (DISC '99),* pp. 49-63, P. Jayanti ed. Sept. 1999.

[18] A. Mostéfaoui and M. Raynal, "Consensus Based on Failure Detectors with a Perpetual Weak Accuracy Property," *Proc. Int'l IEEE Parallel and Distributed Processing Symp. (IPDPS '00), (14th IPPS/11th SPDP),* pp. 514-519, May 2000.

[19] A. Mostéfaoui, M. Raynal, and F. Tronel, "The Best of Both Worlds: a Hybrid Approach to Solve Consensus," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '00, previously FTCS),* pp. 513-522, June 2000.

[20] R. Oliveira, R. Guerraoui, and S. Schiper, "Consensus in the Crash-Recovery Model," Research Report 97-239, Dpartement d'informatique, EPFL, Lausanne, Aug. 1997.

[21] L. Rodrigues and P. Verìssimo, "Topology-Aware Algorithms for Large Scale Communication," *Advances in Distributed Systems,* pp. 1217-1256, 2000.

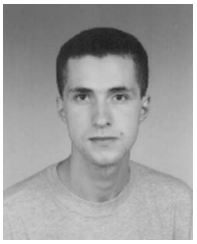
[22] B. Sanders, "The Information Structure of Distributed Mutual Exclusion Algorithms," *ACM Trans. Computer Systems,* vol. 5, no. 3, pp. 284-299, 1987.

[23] A. Schiper, "Early Consensus in an Asynchronous System with a Weak Failure Detector," *Distributed Computing,* vol. 10, pp. 149-157, 1997.

[24] D. Skeen, "Non-Blocking Commit Protocols," *Proc. ACM-SIGMOD Int'l Conf. Management of Data,* pp. 133-142, 1981.

[25] J. Yang, G. Neiger, and E. Gafni, "Structured Derivations of Consensus Algorithms for Failure Detectors," *Proc. 17th ACM Symp. Principles of Distributed Computing.* pp. 297-308, 1998.

Michel Hurfin received the PhD degree in computer science from the University of Rennes. His dissertation topic addressed the problems of execution replay and property detection in distributed applications. In 1994, he spent a postdoctoral year at Kansas State University in the research group of Professor Mizuno. Currently, he conducts his scientific research activities at the INRIA laboratory located in Rennes, Brittany, France. His research interests include distributed systems, software engineering, and middleware for distributed operating systems. Recently, he has initiated research on distributed fault-tolerance middleware.



Achour Mostéfaoui received the engineer degree in computer science in 1990, from the University of Algiers (USTHB) and the PhD degree in computer science in 1994, from the University of Rennes. He is currently an assistant professor in the Computer Science Department of the University of Rennes, France. His research interests include fault tolerance in distributed systems, group communication, consistency in DSM systems, and checkpointing distributed computations. He has published more than 30 scientific publications and served as a reviewer for more than 20 major journals and conferences.

Michel Raynal received the "doctorat d'Etat" in computer science from the University of Rennes, France, in 1981. In 1981, he moved to Brest (France) where he founded the Computer Science Department in a telecommunications engineer school (ENST). In 1984, he moved to the University of Rennes where he has been a professor of computer science. At IRISA (CNRS-INRIA-University joint computing research laboratory located in Rennes), he is the founder and the leader of the research group "Algorithmes distribués et protocoles" (Distributed Algorithms and Protocols). His research interests include distributed algorithms, operating systems, computing systems, protocols, and dependability. His main interest lies in the fundamental principles that underlie the design and the construction of distributed systems. He has been the principal investigator of a number of research grants in these areas (grants from French private companies, France Telecom, CNRS, NFS-INRIA, and ESPRIT). Dr. Raynal has been invited by many universities to give lectures about operating systems and distributed algorithms in Europe, South and North America, Asia, and Africa. He is serving as an editor for two international journals. Dr. Raynal has published more than 70 papers in journals and more than 140 papers in conferences. He has has written seven books devoted to parallelism, distributed algorithms and systems. Among them: *Distributed Computations and Networks*, (MIT Press, 1988) and *Synchronization and Control of Distributed Programs*, coauthored with Jean-Michel Helary, (Wiley, 1990). He has been invited to serve in program committees for more than 50 international conferences. In 1989 and 1995 he served as the program chair of the WDAG/DISC symposium on Distributed Algorithms. In 1995 he was the program chair of the IEEE Conference on Future Trends of Distributed Computing Systems (FT DCS '95). In 1999 he served as the general chair of the IEEE Conference on Object-Oriented Real-time Distributed Computing (ISORC '99) that was held in Saint-Malo, France. He will be the program cochair of IEEE International Conference on Distributed Computing Systems (ICDCS '02) that will be held in Vienna, Austria, and the conference cochair of ISORC '02 that will be held in Washington DC.