Eventually Consistent Failure Detectors*

Mikel Larrea
Universidad del País Vasco
20018 Donostia, Spain
mikel.larrea@si.ehu.es

Antonio Fernández
Universidad Rey Juan Carlos
28933 Móstoles, Spain
afernandez@acm.org

Sergio Arévalo
Universidad Rey Juan Carlos
28933 Móstoles, Spain
s.arevalo@escet.urjc.es

Abstract

The concept of unreliable failure detector *was introduced by Chandra and Toueg [4] as a mechanism that provides information about process failures. This mechanism has been used to solve different problems in asynchronous systems, in particular the Consensus problem.*

In this paper, we present a new class of unreliable failure detectors, which we call Eventually Consistent *and denote by $\diamond\mathcal{C}$. This class adds to the failure detection capabilities of other classes an eventual leader election capability. This capability allows all correct processes to eventually choose the same correct process as leader. We study the relationship between $\diamond\mathcal{C}$ and other classes of failure detectors. We also propose an efficient algorithm to transform $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$ in models of partial synchrony. Finally, to show the power of this new class of failure detectors, we present a Consensus algorithm based on $\diamond\mathcal{C}$. This algorithm successfully exploits the leader election capability of the failure detector, and performs better in number of rounds than all the previously proposed algorithms for failure detectors with eventual accuracy.*

1. Introduction

1.1. Unreliable Failure Detectors

An *unreliable failure detector* is a mechanism that provides (possibly incorrect) information about faulty processes. When it is queried, the failure detector returns a set of processes believed to have crashed (suspected processes). The concept of unreliable failure detector was introduced by Chandra and Toueg in [4], where they proposed several classes of unreliable failure detectors. After this seminal work, a number of authors have proposed other classes of unreliable failure detectors [1, 2, 8, 16]. These failure detectors have been used to solve several fundamental problems in asynchronous distributed systems, e.g., Consensus [4].

In [4], failure detectors were characterized in terms of two properties: *completeness* and *accuracy*. Completeness characterizes the failure detector capability of suspecting every incorrect process (processes that actually crash), while accuracy characterizes the failure detector capability of not suspecting correct processes. Two kinds of completeness and four kinds of accuracy were defined in [4], which combined yield eight classes of failure detectors.

In this paper, we focus on the following completeness and accuracy properties, from those defined in [4]:

- *Strong Completeness.* Eventually every process that crashes is permanently suspected by *every* correct process.
- *Weak Completeness.* Eventually every process that crashes is permanently suspected by *some* correct process.
- *Eventual Strong Accuracy.* There is a time after which correct processes are not suspected by any correct process.
- *Eventual Weak Accuracy.* There is a time after which some correct process is never suspected by any correct process.

Combining in pairs these completeness and accuracy properties, we obtain four different failure detector classes, which are shown in Figure 1.

In [3], Chandra et al. defined a new class of failure detectors, denoted Ω, and used it to prove that $\diamond\mathcal{W}$ is the weakest failure detector class for solving Consensus[1]. The output of the failure detector module of Ω at

*Research partially supported by the Spanish Research Council, contracts TIC99-0280-C02-02, TEL99-0582, and TIC98-1032-C03-01, and the Madrid Regional Research Council, contract CAM-07T/00112/1998.

[1] Actually, to prove their result Chandra et al. show first that Ω is at least as strong as $\diamond\mathcal{W}$, and then that any failure detector $\mathcal{D}$ that can be

	Eventual Strong Accuracy	Eventual Weak Accuracy
Strong Completeness	*Eventually Perfect* $\diamond\mathcal{P}$	*Eventually Strong* $\diamond\mathcal{S}$
Weak Completeness	*Eventually Quasi-Perfect* $\diamond\mathcal{Q}$	*Eventually Weak* $\diamond\mathcal{W}$

Figure 1. Four classes of failure detectors defined in terms of completeness and accuracy.

a process p is a *single* process q, that p currently considers to be *correct* (we say that p *trusts* q). The failure detector Ω satisfies the following property:

Property 1 *There is a time after which all the correct processes always trust the same correct process.*

As with $\diamond\mathcal{W}$, the output of the failure detector module of Ω at a process p may change with time, i.e., p may trust different processes at different times. Furthermore, at any given time t, two processes p and q may trust different processes.

1.2. Consensus Algorithms

The Consensus problem [18] is a fundamental problem in distributed systems. Chandra and Toueg showed in [4] that their unreliable failure detectors allow to solve Consensus in asynchronous systems. This was shown to be impossible in a pure asynchronous system by Fischer et al. [7]. Since then, several distributed fault-tolerant algorithms to solve Consensus based on unreliable failure detectors have been proposed [9, 15, 19].

All the Consensus algorithms based on failure detectors with eventual accuracy we know of require at least a failure detector of class $\diamond\mathcal{S}$. All of them proceed in rounds, in each of which a different process acts as coordinator. This approach is known as the *rotating coordinator paradigm*. If the coordinator of a round crashes or is suspected by several processes the round may fail and the consensus is not reached in that round. With a failure detector of class $\diamond\mathcal{S}$ it is guaranteed the existence of a correct process, namely $leader\_process$, that is eventually not suspected by any correct process. If eventually $leader\_process$ becomes the coordinator after no process suspects it, the consensus is guaranteed to be reached.

The inconvenience of the $\diamond\mathcal{S}$-Consensus algorithms based on the rotating coordinator paradigm is that if $leader\_process$ is not coordinator until round i (where i could be $\Omega(n)$, with n the number of processes), they require i rounds to reach consensus. It would be nice to have Consensus algorithms that quickly choose $leader\_process$ as the coordinator, hence reducing the number of rounds required to reach consensus.

There have been other approaches to solve the Consensus problem in non-synchronous systems. In [6], partially synchronous models are assumed and Consensus algorithms for these models have been proposed. Again, these algorithms use the rotating coordinator paradigm and can present the above problem.

To our knowledge, the first Consensus algorithm that uses a kind of leader election algorithm to choose the coordinator of a round instead of a rotating coordinator is the Paxos Consensus algorithm [10]. In this algorithm a system model different from the above is used, in which periods of synchrony and asynchrony alternate. The Paxos Consensus algorithm proceeds in asynchronous rounds, with a coordinator, given by the leader election algorithm, for each round.

Very recently, in [17] it is proposed a Consensus algorithm based on a failure detector of class Ω that does not use the rotating coordinator paradigm.

1.3. Our Results

In this paper, we define a new accuracy property, which combined with strong completeness defines a new class of unreliable failure detectors, which we call *Eventually Consistent* and denote by $\diamond\mathcal{C}$. The main property of all the failure detectors in this class is that, in each run, all the correct processes eventually converge to the same non-suspected correct process (by means of a deterministic $leader$ function applied to the set of non-suspected processes[2] returned by the failure detector).

The interest of the failure detectors in this class comes from the fact that, implicitly, they provide something like a leader election mechanism. (However, they do not give knowledge of when the leader has been elected and allow several leaders at the same time.) If each process applies the same deterministic function $leader$ to the set of non-suspected processes to choose one from it, eventually all correct processes will permanently agree on the same correct process. If we call this process $leader\_process$, eventually all processes identify $leader\_process$ as their distinguished correct process.

Considering the Consensus problem, this property of the eventually consistent failure detectors allows every correct process to eventually agree on a coordinator that

used to solve Consensus is at least as strong as Ω (and hence at least as strong as $\diamond\mathcal{W}$).

[2] If the failure detector returns a set of suspected processes, this is just the complement of that set.

can be used to reach consensus. Hence, with these failure detectors we do not need to rely on the rotating coordinator paradigm to eventually choose an appropriate coordinator.

We study the relationship between $\diamond\mathcal{C}$ and the four classes of failure detectors satisfying eventual accuracy presented in Figure 1. We first show that $\diamond\mathcal{P}$ is a subclass of $\diamond\mathcal{C}$, i.e., any algorithm implementing $\diamond\mathcal{P}$ implements also $\diamond\mathcal{C}$. Similarly, we show that $\diamond\mathcal{C}$ is a subclass of $\diamond\mathcal{S}$ (and hence of $\diamond\mathcal{W}$). Moreover, we show that classes $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$ are equivalent. This means that we can obtain a failure detector of class $\diamond\mathcal{C}$ from any failure detector of class $\diamond\mathcal{S}$.

We then show that $\diamond\mathcal{C}$ can be implemented as efficiently as $\diamond\mathcal{S}$. To show it, we present some algorithms implementing $\diamond\mathcal{S}$ that *directly* implement $\diamond\mathcal{C}$, i.e., we do not need to execute any transformation algorithm. In particular, the algorithms implementing $\diamond\mathcal{S}$ presented in [12, 14] implement also a failure detector of class $\diamond\mathcal{C}$. We also propose an efficient algorithm transforming $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$ in models of partial synchrony [4, 6].

Finally, to show the power of this class of failure detectors, we present a Consensus algorithm based on $\diamond\mathcal{C}$. This algorithm proceeds in asynchronous rounds and each round is divided in several phases, like most previous $\diamond\mathcal{S}$-Consensus algorithms [9, 15, 19]. The main difference between our algorithm and the above ones is the way the coordinator is selected. We do not use the rotating coordinator paradigm as they do, but the leader election capability of $\diamond\mathcal{C}$.

Compared with the Paxos Consensus algorithm mentioned above, our algorithm works in an asynchronous system extended with a failure detector, while Paxos assumes a system model in which there are periods of synchrony. In fact, the leader election algorithms (which is basically a failure detection algorithm) strongly relies on the existence of periods of synchrony in the system. Besides that, both algorithms use similar approaches. The Consensus algorithm of [17] is based on the same ideas as ours. In fact, they refer to preliminary versions of this paper [11, 13].

The rest of the paper is organized as follows. In Section 2, we establish the model of the system we use in the rest of the paper and define the new class of failure detectors ($\diamond\mathcal{C}$). In Section 3, we study its relationship with other classes of failure detectors. In Section 4, we present an efficient Consensus algorithm based on $\diamond\mathcal{C}$. Finally, Section 5 concludes the paper.

2. Definitions

2.1. System Model

We consider a distributed system consisting of a finite totally ordered set Π of n processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$. Processes communicate only by sending and receiving messages. Every pair of processes is assumed to be connected by a reliable communication channel. The system is *asynchronous*, i.e., there are no timing assumptions about neither the relative speeds of the processes nor the delay of messages. Processes can fail by *crashing*, that is, by prematurely halting. Crashes are permanent, i.e., crashed processes do not recover.

A *distributed failure detector* can be viewed as a set of n failure detection modules, each one attached to a different process in the system. These modules cooperate to satisfy the required properties of the failure detector. Upon request, each module provides its attached process with a set of processes it suspects to have crashed. These sets can differ from one module to another at a given time. Let us denote by $\mathcal{D}_p$ the set of suspected processes returned by a failure detector $\mathcal{D}$ to a given process p. We also denote by $\mathcal{T}_p$ the set of trusted (non-suspected) processes of the failure detection module attached to process p, i.e., $\mathcal{T}_p = \Pi - \mathcal{D}_p$. We assume that a process interacts only with its local failure detection module in order to get the current set of suspected processes.

2.2. Eventually Consistent Failure Detectors

We introduce now the class of eventually consistent failure detectors. The main characteristic of these failure detectors is the accuracy property they satisfy, which we call *Eventual Consistent Accuracy*. Informally, the eventual consistent accuracy guarantees that there is a correct process p that is eventually and permanently not suspected by any correct process, and that there is a function that each correct process can apply to the output of its local failure detection module that eventually and permanently returns p.

More formally, the eventual consistent accuracy property can be defined as follows. Let $\mathcal{P}(\Pi)$ be the power set of the set Π.

Definition 1 *A failure detector $\mathcal{D}$ satisfies* Eventual Consistent Accuracy *if there is a deterministic function $leader : \mathcal{P}(\Pi) \rightarrow \Pi$, a time t and a correct process p such that, after t, for every correct process q, $p \notin \mathcal{D}_q$ and $leader(\mathcal{T}_q) = p$.*

Definition 2 *We define the* Eventually Consistent *class of failure detectors, denoted $\diamond\mathcal{C}$, as those that satisfy*

both the strong completeness and the eventual consistent accuracy properties.

A failure detector of class $\diamond\mathcal{C}$ enhances the classical failure detection properties of previously defined classes with an eventual leader election mechanism. These failure detectors guarantee that after some point in time all correct processes can behave as a consistent leader election algorithm. (However, they do not give knowledge of when the leader has been elected, and allow several leaders at the same time.) This property can be used by algorithms in which the safety properties are not affected by the simultaneous existence of several leaders, and that guarantee termination if a unique leader exists. Furthermore, these failure detectors can be very useful to algorithms that have early termination when there is a unique leader. As it is well known, usually it is not necessary for the failure detector to reach permanent stability to be useful. Instead, many algorithms can successfully complete if the failure detector is stable (provides a unique leader) for long enough periods of time.

3. Relation between $\diamond\mathcal{C}$ and other Failure Detector Classes

With an eventually consistent failure detector, from the eventual consistent accuracy, eventually there is a correct process that will be never suspected by any correct process, and from the strong completeness, eventually all the crashed processes are permanently suspected by every correct process. From this, it is simple to see that every failure detector of class $\diamond\mathcal{C}$ belongs also to the class $\diamond\mathcal{S}$ (and hence to $\diamond\mathcal{W}$).

Note also that any failure detector of class $\diamond\mathcal{P}$ belongs to the class $\diamond\mathcal{C}$ as well. With $\diamond\mathcal{P}$, eventually the set of suspected processes by every correct process becomes the same, containing only all the processes that actually crash. A possible $leader$ function could always choose the first (with respect to the order $p_1, \ldots, p_n$ assumed in the system model) non-suspected process. Figure 2 summarizes the relation between failure detector classes $\diamond\mathcal{W}$, $\diamond\mathcal{S}$, $\diamond\mathcal{C}$, and $\diamond\mathcal{P}$.

3.1. Equivalence of $\diamond\mathcal{C}$ and Ω

It is straightforward to show that failure detector classes $\diamond\mathcal{C}$ and Ω are equivalent, i.e., one can be transformed into the other and vice versa. In some sense, the class $\diamond\mathcal{C}$ can be viewed as a redefinition of Ω in terms of lists of suspected processes instead of a single trusted process. Nevertheless, the class $\diamond\mathcal{C}$ does not force each failure detector module to systematically suspect all the processes in the system except one, while this is implicitly done by Ω. The following lemma shows the equivalence of Ω and $\diamond\mathcal{C}$.

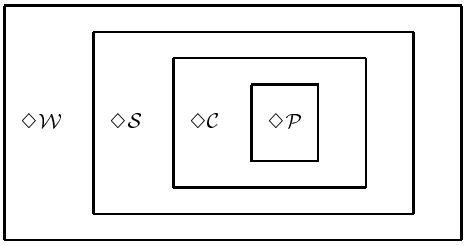

Figure 2. Relation between failure detector classes $\diamond\mathcal{W}$, $\diamond\mathcal{S}$, $\diamond\mathcal{C}$ and $\diamond\mathcal{P}$.

Lemma 1 *Failure detector classes Ω and $\diamond\mathcal{C}$ are equivalent.*

Proof: To transform Ω into $\diamond\mathcal{C}$: every process p queries its local failure detection module Ω_p and suspects every process except its trusted process. To transform $\diamond\mathcal{C}$ into Ω: every process p queries its local failure detection module $\diamond\mathcal{C}_p$ and applies the $leader$ function to the complementary set $\Pi - \diamond\mathcal{C}_p$. The resulting process is output by p as its trusted process. ■

3.2. Equivalence of $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$

We now show that any failure detector of class $\diamond\mathcal{S}$ can be transformed into a failure detector of class $\diamond\mathcal{C}$. Since, by definition, $\diamond\mathcal{C}$ is a subclass of $\diamond\mathcal{S}$, i.e., every failure detector in $\diamond\mathcal{C}$ is in $\diamond\mathcal{S}$, this shows that failure detector classes $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$ are actually equivalent.

Lemma 2 ([4]) *Failure detector classes $\diamond\mathcal{S}$ and $\diamond\mathcal{W}$ are equivalent.*

Lemma 3 ([3]) *Failure detector classes $\diamond\mathcal{W}$ and Ω are equivalent.*

Theorem 1 *Failure detector classes $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$ are equivalent.*

Proof: To transform $\diamond\mathcal{C}$ into $\diamond\mathcal{S}$: trivial, since $\diamond\mathcal{C}$ is a subclass of $\diamond\mathcal{S}$ (both classes satisfy strong completeness, and eventual consistent accuracy ($\diamond\mathcal{C}$) involves eventual weak accuracy ($\diamond\mathcal{S}$)). To transform $\diamond\mathcal{S}$ into $\diamond\mathcal{C}$: follows from Lemmas 1, 2, and 3. ■

Thus, having any failure detector of class $\diamond\mathcal{S}$, it is always possible to build a failure detector of class $\diamond\mathcal{C}$.

However, instead of starting from any failure detector of class $\diamond\mathcal{S}$ and running a – maybe expensive[3] – transformation protocol on top of it in order to get a failure detector of class $\diamond\mathcal{C}$, we show in the next section that there are extremely efficient implementations of $\diamond\mathcal{C}$.

3.3. Implementations of $\diamond\mathcal{C}$

There are several algorithms implementing failure detectors of the classes presented in Figure 1 that also implement an eventually consistent failure detector. For instance, since $\diamond\mathcal{P}$ is a subclass of $\diamond\mathcal{C}$, the $\diamond\mathcal{P}$ algorithms of Chandra and Toueg [4] and Larrea et al. [12], implement also a failure detector of class $\diamond\mathcal{C}$.

Concerning the ring-based algorithm implementing $\diamond\mathcal{S}$ proposed in [12], the set of non-suspected processes can be different in different processes, but the algorithm guarantees that eventually the first (starting from the *initial candidate to leader* and following the order defined by the ring) non-suspected process is the same for every correct process, and that it is correct. From this, it is easy to derive a deterministic function $leader$ that returns the first non-suspected process. Hence, the ring-based algorithm implementing $\diamond\mathcal{S}$ of [12] implements also a failure detector of class $\diamond\mathcal{C}$.

In [14], an even more efficient algorithm implementing a failure detector of class $\diamond\mathcal{S}$ is proposed. In this case, the set of non-suspected processes contains only one process, that will eventually and permanently be the same correct process for all correct processes, which trivially gives us a possible $leader$ function. Again, this algorithm implements also a failure detector of class $\diamond\mathcal{C}$. With this algorithm, in stability, the number of messages periodically sent is $n-1$.

3.4. Transforming $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$

In this section, we present an efficient algorithm that transforms any failure detector of class $\diamond\mathcal{C}$ into a failure detector of class $\diamond\mathcal{P}$ in models of partial synchrony [4, 6]. The approach followed is to use the eventually agreed trusted process to build and propagate a list of suspected processes that satisfies the properties of $\diamond\mathcal{P}$.

Figure 3 presents the algorithm in detail, which works as follows. Each *leader* process (i.e., each process that considers itself as leader by consulting its failure detection module) builds a local list of suspected processes by using time-outs (Tasks 3 and 4), and sends its list periodically to the rest of processes (Task 1). Concurrently, each non-leader process periodically sends an I-AM-ALIVE message to its leader process (Task 2). Finally, when a process receives a list of suspected processes from its leader process, it adopts this list as its own list (Task 5).

If we assume that most of the time the failure detector provides a unique leader, then the cost of this transformation algorithm in terms of the number of messages periodically sent is $2(n-1)$, since the leader process sends a message to the rest of processes, and every non-leader process sends a message to the leader process. Thus, the algorithm has a linear cost. Furthermore, this cost can be reduced in practice. If we assume that the algorithm implementing $\diamond\mathcal{C}$ requires the leader process to periodically send a message to the rest of processes (this is the case of the algorithm proposed in [14]), then the list of suspected processes can be piggybacked on this message, reducing the number of messages of the transformation algorithm to the half.

Every process p executes the following:

$suspected_p \leftarrow \emptyset$ {$suspected_p$ *provides the properties of* $\diamond\mathcal{P}$}
for all $q \in \Pi$ {$\Delta_p(q)$ *denotes the duration of* p*'s time-out for* q}
  $\Delta_p(q) \leftarrow$ default time-out interval

cobegin

|| *Task 1:* **repeat periodically**
  if $leader(\mathcal{T}_p) = p$ **then**
    send $suspected_p$ to the rest of processes

|| *Task 2:* **repeat periodically**
  if $leader(\mathcal{T}_p) \neq p$ **then**
    send "p-is-alive" to $leader(\mathcal{T}_p)$

|| *Task 3:* **repeat periodically**
  if $leader(\mathcal{T}_p) = p$ **then**
    for all $q \in \Pi, q \neq p$:
      if $q \notin suspected_p$ and p did not receive "q-is-alive"
        during the last $\Delta_p(q)$ ticks of p's clock **then**
        $suspected_p \leftarrow suspected_p \cup \{q\}$
            {p *times-out on* q*: it suspects* q *has crashed*}

|| *Task 4:* **when** receive "q-is-alive" for some q
  if $leader(\mathcal{T}_p) = p$ **and** $q \in suspected_p$ **then**
            {p *knows that it prematurely timed-out on* q}
    $suspected_p \leftarrow suspected_p - \{q\}$ {*1.* p *repents on* q*, and*}
    $\Delta_p(q) \leftarrow \Delta_p(q) + 1$ {*2.* p *increases its time-out for* q}

|| *Task 5:* **when** receive $suspected_q$ for some q
  if $leader(\mathcal{T}_p) = q$ **then**
    $suspected_p \leftarrow suspected_q$ {p *adopts* $suspected_q$}

coend

Figure 3. Transforming $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$ in models of partial synchrony.

[3] The transformation protocols of Chandra et al. [3] and Chu [5] are quadratic, i.e., they require that every process sends messages periodically to all processes in the system.

Following the previous strategy, we get an extremely efficient implementation of $\diamond\mathcal{P}$ that has a cost of $2(n-1)$ messages periodically sent ($n-1$ of the implementation of $\diamond\mathcal{C}$ proposed in [14], and $n-1$ of the transformation algorithm of Figure 3). This compares favorably to the implementation of $\diamond\mathcal{P}$ proposed by Chandra and Toueg in [4], which has a cost of n^2. Also, this cost is the same as that of the ring algorithm implementing $\diamond\mathcal{P}$ proposed by Larrea et al. in [12], but this approach has the additional benefit of not suffering of the high latency in crash detection of this algorithm (due to the propagation of the list of suspected processes over the ring).

4. Solving Consensus using $\diamond\mathcal{C}$

In this section, we present an algorithm that solves Uniform Consensus using an eventually consistent failure detector. We assume the system model defined in Section 2. In addition, we assume that the system is augmented with a failure detector $\mathcal{D}$ of class $\diamond\mathcal{C}$, to which processes have access. We also assume that all processes know the function $leader$ associated with the failure detector, as specified in Definition 1. Finally, we assume that a majority of processes are correct, i.e., do not crash. Thus, if we denote by f the number of processes that can fail, we assume $f < n/2$. This is a necessary requirement to solve Consensus using $\diamond\mathcal{C}$ in asynchronous systems. This can be derived from the fact that $\diamond\mathcal{P}$ is a subclass of $\diamond\mathcal{C}$, and Theorem 6.3.1 in [4], which claims that to solve Consensus with $\diamond\mathcal{P}$ there must be a majority of correct processes.

Figures 4 and 5 present the algorithm in detail. Each process runs an instance of this algorithm, which proceeds in asynchronous rounds. As the $\diamond\mathcal{S}$-Consensus algorithm of Chandra and Toueg [4], it goes through three asynchronous epochs, each of which may span several rounds. In the first epoch, several decision values are possible. In the second epoch, a value gets *locked*: no other decision value is possible. In the third epoch, processes decide the locked value.

Each round of the algorithm is divided into five asynchronous phases. In Phase 0, every process determines its coordinator for the round. A process becomes its own coordinator for the round if it is the process returned by the function $leader$. A coordinator announces itself by sending a message to the rest of processes. A process becomes a non-coordinator, i.e., a participant, if it first receives a message from a coordinator, which becomes its coordinator for the round. In Phase 1, every process sends its current estimate of the decision value time-stamped with the round number in which it adopted this estimate, to its coordinator. Also after phase 0 and concurrently with the main algorithm, each process sends a *null* estimate to the other coordinators of the round (first task of Figure 5).

In Phase 2, each coordinator tries to gather a majority of estimates. If it succeeds, then it selects an estimate with the largest time-stamp and sends it to all the processes as a proposition. On the other hand, if it does not receive a majority of estimates then it sends a *null* proposition to all processes. In Phase 3, each process waits for a proposition from its coordinator. However, it also stops waiting if it suspects its coordinator or if it receives a non-null proposition from some other coordinator. If the process receives a non-null proposition from some coordinator (including its own), then it adopts it and sends an ack message to this coordinator. If the process receives a *null* proposition from its coordinator, it stops waiting and passes to the next phase. Finally, if the process suspects its coordinator, it sends a $nack$ message to it. After this phase and concurrently with the main algorithm, each process sends a $nack$ message to a coordinator from which it receives a non-null proposition for this round (second task of Figure 5). Finally, in Phase 4 the coordinator that succeeded in Phase 2 and sent a non-null proposition (if any, and as we will see at most one) waits for a majority of ack/$nack$ messages. If it gathers a majority of ack messages, then it knows that a majority of processes adopted its proposition as their new estimate. Consequently, this coordinator R-broadcasts a request to decide its proposition. At any time, if a process R-delivers such a request, it decides accordingly.

Note that this Consensus algorithm does not use the rotating coordinator paradigm. Instead, the eventual leader election functionality provided by the failure detector is exploited. As a result, in the case of stability of the failure detector (i.e., the $leader$ function returns the same correct process to all processes), Consensus is solved in only one round, providing early consensus. In any $\diamond\mathcal{S}$-Consensus algorithm based on the rotating coordinator paradigm, the number of rounds can be $\Omega(n)$ once the failure detector is stable (i.e., there is one correct process that is never suspected by any process), until a correct and not-suspected process becomes coordinator of a round.

5. Conclusions

In this paper, we have proposed a novel class of unreliable failure detectors, called Eventually Consistent and denoted $\diamond\mathcal{C}$. We have studied the relationship between $\diamond\mathcal{C}$ and other failure detector classes, showing that $\diamond\mathcal{P}$ is a subclass of $\diamond\mathcal{C}$, and that $\diamond\mathcal{C}$ is a subclass of $\diamond\mathcal{S}$. Moreover, we have shown that $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$ are equivalent classes. We have presented two algorithms imple-

Every process p executes the following:

procedure *propose*(v_p)
  $estimate_p \leftarrow v_p$ {$estimate_p$ *is p's estimate of the decision value*}
  $state_p \leftarrow undecided$
  $r_p \leftarrow 0$ {r_p *is p's current round number*}
  $ts_p \leftarrow 0$ {ts_p *is the last round in which p updated* $estimate_p$*, initially 0*}

  while $state_p = undecided$ {*Rotate until decision is reached*}
    $chosen_p \leftarrow false$
    $replied_p \leftarrow false$
    $r_p \leftarrow r_p + 1$

    Phase 0: {*Each process determines its coordinator for the round*}
      wait until [$p = leader(\mathcal{T}_p)$ **or** for a process q: received $(q, r_p, coordinator)$] {*Query the failure detector*}
      if [for a process q: received $(q, r_p, coordinator)$] **then**
        $c_p \leftarrow q$
      else
        $c_p \leftarrow p$
        send $(p, r_p, coordinator)$ to all processes except p
      $chosen_p \leftarrow true$

    Phase 1: {*Each process p sends* $estimate_p$ *to its current coordinator*}
      send $(p, r_p, estimate_p, ts_p)$ to c_p

    Phase 2: {*Each coordinator tries to gather* $\lceil \frac{(n+1)}{2} \rceil$ *estimates to propose a new estimate*}
      if $p = c_p$ **then**
        wait until [for $\lceil \frac{(n+1)}{2} \rceil$ processes q: received $(q, r_p, estimate_q, ts_q)$ **or** $(q, r_p, null\_estimate, 0)$]
        $msgs_p[r_p] \leftarrow \{(q, r_p, estimate_q, ts_q) \mid p$ received $(q, r_p, estimate_q, ts_q)$ from $q\}$
        if [for $\lceil \frac{(n+1)}{2} \rceil$ processes q: received $(q, r_p, estimate_q, ts_q)$] **then**
          $decidible_p \leftarrow true$
          $t \leftarrow$ largest ts_q such that $(q, r_p, estimate_q, ts_q) \in msgs_p[r_p]$
          $estimate_p \leftarrow$ select one $estimate_q$ such that $(q, r_p, estimate_q, t) \in msgs_p[r_p]$
          send $(p, r_p, estimate_p)$ to all
        else {*p received null_estimate from some process*}
          $decidible_p \leftarrow false$
          send $(p, r_p, null\_estimate)$ to all

    Phase 3: { *Each process waits for a new estimate proposed by a coordinator or to receive null_estimate from its coordinator or to suspect it* }
      wait until [for a process q: received $(q, r_p, estimate_q)$ **or** received $(c_p, r_p, null\_estimate)$ from c_p **or** $c_p \in \mathcal{D}_p$]
      if [for a process q: received $(q, r_p, estimate_q)$] **then** {*p received* $estimate_q$ *from a process q*}
        $estimate_p \leftarrow estimate_q$
        $ts_p \leftarrow r_p$
        send (p, r_p, ack) to q
      else if [received $(c_p, r_p, null\_estimate)$ from c_p] **then** {*p received* $null\_estimate$ *from* c_p}
        discard message
      else {*p suspects that* c_p *crashed*}
        send $(p, r_p, nack)$ to c_p
      $replied_p \leftarrow true$

    Phase 4: { *The coordinator that can still decide (if any) waits for* $\lceil \frac{(n+1)}{2} \rceil$ *replies. If they indicate that* $\lceil \frac{(n+1)}{2} \rceil$ *processes adopted its estimate, the coordinator R-broadcasts a decide message* }
      if $(p = c_p)$ **and** $(decidible_p)$ **then**
        wait until [for $\lceil \frac{(n+1)}{2} \rceil$ processes q: received (q, r_p, ack) **or** $(q, r_p, nack)$]
        if [for $\lceil \frac{(n+1)}{2} \rceil$ processes q: received (q, r_p, ack)] **then**
          R-broadcast$(p, r_p, estimate_p, decide)$

Figure 4. Solving Consensus using any $\mathcal{D} \in \diamond\mathcal{C}$.

when received $(q, r_q, coordinator)$ from q such that $(r_q < r_p)$ **or**
$((r_q = r_p)$ **and** $(chosen_p))$
send $(p, r_q, null\_estimate, 0)$ to q

when received $(q, r_q, estimate_q)$ from q such that $(r_q < r_p)$ **or**
$((r_q = r_p)$ **and** $(replied_p))$
send $(p, r_q, nack)$ to q

{*If p R-delivers a decide message, p decides accordingly*}
when *R-deliver*$(q, r_q, estimate_q, decide)$
if $state_p = undecided$ **then**
decide$(estimate_q)$
$state_p \leftarrow decided$

Figure 5. Separate tasks for replying to late coordinators and taking the decision.

menting a failure detector of class $\diamond\mathcal{S}$ that also implement a failure detector of class $\diamond\mathcal{C}$, showing that $\diamond\mathcal{C}$ can be implemented as efficiently as $\diamond\mathcal{S}$. We have also proposed an efficient algorithm transforming $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$ in models of partial synchrony.

On top of their failure detection capability, the failure detectors of class $\diamond\mathcal{C}$ have an eventual leader election functionality. This property can be very useful. To demonstrate it, we have presented an efficient algorithm for solving Consensus based on an eventually consistent failure detector. The class $\diamond\mathcal{C}$ allows the algorithm to use a more selective approach to choose a coordinator. This approach allows our algorithm to reach consensus in one single round in stability, while $\diamond\mathcal{S}$-Consensus algorithms based on the rotating coordinator paradigm may require $\Omega(n)$ rounds.

Acknowledgments. We are grateful to André Schiper for his valuable comments.

References

[1] M. Aguilera, W. Chen, and S. Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG)*. LNCS, Springer-Verlag, September 1997.

[2] M. Aguilera, S. Toueg, and B. Deianov. Revisiting the weakest failure detector for uniform reliable broadcast. In *Proceedings of the 13th International Symposium on DIstributed Computing (DISC, formerly WDAG)*, pages 19–33. LNCS, Springer-Verlag, September 1999.

[3] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[5] F. Chu. Reducing Ω to $\diamond\mathcal{W}$. *Information Processing Letters*, 67:289–293, 1998.

[6] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[7] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[8] R. Guerraoui and A. Schiper. Γ-accurate failure detectors. In *Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG)*, pages 269–286. LNCS, Springer-Verlag, October 1996.

[9] M. Hurfin and M. Raynal. A simple and fast asynchronous consensus protocol based on a weak failure detector. *Distributed Computing*, 12(4):209–223, 1999.

[10] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[11] M. Larrea. *Efficient Algorithms to Implement Failure Detectors and Solve Consensus in Distributed Systems*. PhD thesis, University of the Basque Country, San Sebastián, October 2000.

[12] M. Larrea, S. Arévalo, and A. Fernández. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *Proceedings of the 13th International Symposium on DIstributed Computing (DISC'99)*, pages 34–48. LNCS, Springer-Verlag, September 1999.

[13] M. Larrea, A. Fernández, and S. Arévalo. Eventually consistent failure detectors. Technical Report, Universidad Pública de Navarra, April 2000. Brief Announcement, 14th International Symposium on Distributed Computing (DISC'2000), Toledo, Spain, October 2000.

[14] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'2000)*, pages 52–59, Nurenberg, Germany, October 2000.

[15] A. Mostefaoui and M. Raynal. Solving consensus using Chandra-Toueg's unreliable failure detectors: a general quorum-based approach. In *Proceedings of the 13th International Symposium on DIstributed Computing (DISC'99)*, pages 49–63. LNCS, Springer-Verlag, September 1999.

[16] A. Mostefaoui and M. Raynal. Unreliable failure detectors with limited scope accuracy and an application to consensus. In *Proceedings of the 19th International Conference on Foundations of Software Technology and Theoretical Computer Science, FST&TCS'99*, pages 329–340. LNCS, Springer-Verlag, December 1999.

[17] A. Mostefaoui and M. Raynal. Leader-based consensus. Technical Report 1372, IRISA, December 2000.

[18] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[19] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, April 1997.