How to Implement a Time-Free Perfect Failure Detector in Partially Synchronous Systems

Gérard Le Lann
INRIA Rocquencourt
Domaine de Voluceau BP 105
F-78153 Le Chesnay Cedex (France)
Email: gerard.le_lann@inria.fr

Ulrich Schmid
Technische Universität Wien
Embedded Computing Systems Group E182/2
Treitlstraße 3, A-1040 Vienna (Austria)
Email: s@ecs.tuwien.ac.at

***Abstract*— This paper[1] shows how to implement a time-free perfect failure detector in a partially synchronous distributed system. Rather than upper and lower bounds on the minimum and maximum end-to-end computation + transmission delays between correct processors, our algorithm needs to know only an upper bound on their ratio. Since this bound may still hold when an assumed bound on the maximum delay is violated, our perfect failure detector may still work correctly in situations where synchronous ones fail, i.e., has higher coverage. We prove that our solution, which employs heartbeat messages and a timer-free "timeout" mechanism based upon synchronized heartbeat rounds, indeed satisfies the properties of a perfect failure detector and can be generalized in several ways.**

I. Introduction

It has been taken for granted for many years that fault-tolerant distributed *real-time* computing problems admit solutions designed in synchronous computational models only. Unfortunately, given the difficulty of ensuring that assumed bounds on computation times and transmission delays are always met (which is notoriously difficult with many systems, especially those built out of COTS products), the safety/liveness/timeliness properties achieved with such systems may have a poor coverage.[2] This holds true for any solution that rests upon (some) timed semantics, including timed asynchronous systems [1] and the Timely Computing Base [2].

Time-free safety and liveness properties can also be guaranteed in purely asynchronous computational models, however. Since asynchronous algorithms do not depend upon timing assumptions, they cannot violate such properties in case the underlying system's assumed timing properties are not met. The coverage of such time-free solutions is hence necessarily higher than that of a solution involving some timing assumptions.

The apparent contradiction between asynchronous algorithms and timeliness properties can be resolved by adhering to the *design immersion* principle, which was introduced in [3] and referred to as the *late binding* principle in [4]. Design immersion permits to consider time-free (pure asynchronous) algorithms for implementing high-level services in real-time systems, by enforcing that timing-related conditions (like "delay for time X") are expressed as time-free logical conditions (like "delay for x round-trips"). Safety and liveness properties can hence be proved *independently* of the timing properties of the system where the asynchronous solution will eventually be run. Timeliness properties are established only late in the design process, by worst-case schedulability analysis, when an asynchronous design solution is immersed in a (necessarily synchronous) real system.

Given that many important problems in fault-tolerant distributed computing do not have deterministic solutions in the pure asynchronous model [5], it must be enriched with some time-free semantics for circumventing impossibility results. This requires solutions for implementing such added semantics, however, these solutions being bound to achieve some very high coverage. The present paper explores the following fundamental theoretical and practical question: Is there a *time-free* solution for implementing a powerful time-free semantics? Such a solution could be proved correct independently of the timing properties of the system where it is eventually run, and just immersed into the system to be fielded as any other asynchronous design.

[1]This paper emanated from a visit of the first named author at TU Vienna, which has been supported by the Austrian START programme Y41-MAT.

[2]The coverage of an assertion is the probability or the likelihood that this assertion holds true in some given universe. Coverage is hence the central issue in critical systems design.

Our paper answers this fundamental question in the affirmative, by providing a time-free implementation of a perfect failure detector $\mathcal{P}$ [6]. It is based upon a new partially synchronous system model, which stipulates a bounded end-to-end computational + transmission delay ratio Θ only. Since our FD algorithm does not incorporate any bound on delays and works even in situations where actual delays are unbounded, our Θ-model effectively allows us to escape from both the impossibility of implementing $\mathcal{P}$ in presence of *unknown* delay bounds of [7] and the impossibility of consensus in presence of *unbounded* delays of [8]. Consequently, the numerous failure detector-based distributed algorithms developed during the last decade can be used for building real-time systems with coverage requirements that cannot be met by existing synchronous solutions.

The remaining sections are organized as follows: Following a detailed relation to existing work in Section II, we present our Θ-model in Section III and discuss coverage issues in Section IV. The basic operation principle of our novel FD solution is outlined in Section V, the detailed algorithm is presented and proved correct in Section VI. The paper concludes with a number of extensions of our approach in Section VII.

II. Problem Description and Related Work

Failure detectors [6] consist of a set of (low level) modules, one per processor[3], which can be queried locally by the higher level processes in order to obtain information about which processors (seem to) have crashed. This information neither needs to be always correct nor consistent at different processors. It must be in accordance with some axiomatic (time-free) FD specification, however, which is usually made up of two parts: *Completeness*, addressing an FD's ability to correctly suspect crashed processors, and *accuracy*, addressing an FD's ability not to incorrectly suspect non-crashed processors.

Several classes of FD specifications have been introduced in [6], which differ primarily in their accuracy requirements. The strongest one is the *perfect failure detector* $\mathcal{P}$, defined by

(SC) *Strong completeness*: Eventually, every processor that crashes is permanently suspected by every correct processor.

(SA) *Strong accuracy*: No processor is suspected before it crashes.

It has been shown in [6] and in the wealth of subsequent work that most problems in fault-tolerant distributed computing, such as consensus, atomic broadcast or leader election, can be solved in systems where $\mathcal{P}$ (even some weaker FD) is implementable. Note that this requires at least a partially synchronous system model, since an implementation of $\mathcal{P}$ in a purely asynchronous system would contradict [5].

Any work on partially synchronous systems we are aware of assumes that delay bounds are either known, see e.g. [9], [10], or are unknown but finite and can hence be learned during the execution via "incremental" timeouts [11]. It has in fact been shown in [8] that consensus—and hence $\mathcal{P}$—cannot be implemented in systems where just one process could crash if either processing or communication delays are unbounded.

The seminal paper [11] classifies partial synchrony according to whether bounds upon the maximum relative processing speeds (Φ) and the maximum absolute communication delays (Δ) exist but are either unknown, or are known but hold only after some unknown *global stabilization time* GST. Those two models were combined into a single generalized partially synchronous model in [6]. It assumes that relative speeds, delays and message losses are arbitrary up to the unknown GST; after GST, no message may be lost and all relative speeds and all communication delays must be smaller than the unknown upper bounds Φ and Δ, respectively. This model underlies most research on failure detectors.

Since it has been shown in [7] that *perpetual* FDs cannot be implemented in partially synchronous systems with unknown delay bounds [6], [11], perpetual accuracy properties like *strong accuracy* ("no processor is suspected before it crashes") are usually replaced by *eventual* ones ("there is a time after which correct processors are not suspected by any correct processor"). Many papers deal with the implementation of such eventual-type FDs [4], [6], [12]–[24]. We note, however, that eventual properties are usually in conflict with the timeliness requiremens of real-time systems.

In [6], a simple implementation of an eventually perfect failure detector $\diamond\mathcal{P}$ for the generalized partially synchronous model was given. It is based upon monitoring periodic "I am alive"-messages using adaptive

[3] In the literature, failure detectors are usually associated with a single process. It is assumed, however, that the crash of a process causes all activities of its failure detector module to cease as well. Given that most failure detector implementations require a dedicated process, e.g. for emitting "I am alive"-messages, we find it appropriate to use the term processor instead of the more common term process in this paper.

(increasing) timeouts at all receiver processors. Starting from an a priori given initial value, the timeout value is increased every time a false suspicion is detected (which occurs when an "I am alive"-message from a suspected processor drops in). By restricting the recipients of "I am alive"-messages from all processors to suitably chosen subsets, a less costly implementation of an eventually strong failure detector $\diamond\mathcal{S}$ was derived in [17].

Alternative FD implementations, which use polling by means of ping/reply roundtrips instead of "I am alive"-messages, have also been proposed for partially synchronous systems. The message-efficient algorithms of [15] use a logical ring, where processors poll only their neighbors and use an adaptive (increasing) timeout for generating suspicions. A similar monitoring technique is used in the adaptive failure detection protocol implementing $\diamond\mathcal{P}$ in timed asynchronous systems [21], which uses piggybacking of FD messages upon application messages in order to reduce the message load.

A major deficiency of most existing adaptive timeout approaches is their inability to also decrease the timeout value, cp. [17]. Obviously, such solutions cannot adapt to (slowly) varying delays over time. Stochastic delay estimation techniques as in [18], [22] could be used if one accepts decreased performance w.r.t. accuracy. QoS aspects—as well as scalability and overall system load—in a probabilistic framework are also addressed in [20].

A different type of unreliable failure detectors that received considerable attention recently [23] is Ω, which outputs just a single—eventually common—processor that is considered up and running. Ω also allows to solve consensus [14] and can be implemented very efficiently even in partially synchronous systems where only some links eventually respect communication delay bounds.

In purely asynchronous systems, it is impossible to implement even eventual-type failure detectors. FDs with very weak specifications [13], [19] have been proposed as an alternative here. The *heartbeat failure detectors* of [19] do not output a list of suspects but rather a list of unbounded counters. Like $\diamond\mathcal{P}$, they permit to solve the important problem of quiescent reliable communication in the presence of crashes, but unlike $\diamond\mathcal{P}$, they can easily be implemented in purely asynchronous systems.

In view of the impossibility results of [8] and [7], it was taken for granted until recently that implementing perpetual failure detectors requires accurate knowledge of delay bounds and hence a synchronous system model. Still, the algorithms presented in [24] reveal that perpetual FDs can be implemented in a time-free manner in systems with specific properties. For example, there is a time-free implementation of $\mathcal{P}$ in systems where every correct processor is connected to a set of $f+1$ processors via links that are not among their f slowest ones. The algorithm cannot verify whether the underlying system actually satisfies this property, however, and thus needs an a priori guarantee for correct FD operation.

In [4], we showed that the problem of implementing perpetual failure detectors in a synchronous system comprising n processors initially can be stated as a generic real-time scheduling problem. A *fast* failure detector algorithm was given as a function of x, which serves to dynamically recompute timeout values. It implements FDs Ω, $\mathcal{S}$, or $\mathcal{P}$, by assigning x value 1, s, or n, respectively, where s is the smallest number of stable processors that cannot be suspected unless they fail. The proposed FD achieves very low detection time [18] by exploiting the fact that failure detection can be separated from the application (i.e., the atop running consensus algorithm). The failure detector service can hence be run as high-priority processes that exchange high-priority FD-level messages, scheduled according to some head-of-the-line policy. As a consequence, such *fast failure detectors* admit an accurate worst-case schedulability analysis, which provides tight bounds on the (inherently small) end-to-end delay of FD-level messages.

III. THE Θ-MODEL

We consider a distributed system of n processors, which are fully interconnected by perfect links. Let δ_{pq} denote the end-to-end computational + transmission delay required for assembly, transmission and processing of any FD-level message sent from some correct processor p to some correct processor q (including $q = p$). Note that δ_{pq} includes any computation of the distributed (failure detector) algorithm at both p and q, including process and message scheduling delays. The following Definition 1 captures the essentials of our Θ-model.

Definition 1 (Θ-Model): Let τ^- resp. τ^+, with uncertainty $\varepsilon = \tau^+ - \tau^-$, be the the minimum resp. maximum end-to-end computational + transmission delay δ_{pq} between any two correct processes p, q, observed during any sufficiently large time window. The system must be such that the delay ratio $\Theta = \tau^+/\tau^-$ respects a given[4] upper bound $\overline{\Theta}$.

Remarks:

1) Note carefully that τ^- and τ^+ need to hold within a small (algorithm-dependent) time window

[4] Overbars are used for assumed bounds ($\overline{\Theta}$) on actual values (Θ).

only. On a larger time-scale, those bounds may be arbitrarily time-variant. In case of the FD of Section VI, the time window is in the order a single round-trip (more specifically, the maximum duration of one instance of consistent broadcasting).

2) Both the perfect link assumption and the fact that τ^- must incorporate "self-reception" ($q = p$) can be avoided by using our powerful perception-based failure model [25], see [26] for details.

It is important to note that the Θ-model is not equivalent to any of the existing partially synchronous models [6], [11]: None of those models can be mapped to the Θ-model, since they all assume $\tau^- = 0$ such that $\Theta = \infty$. The Θ-model cannot be mapped to any model of [6], [11] either: Although those models stipulate a ratio Φ for computational speeds, they also need a (known or unknown) upper bound Δ on *absolute* communication delays. Obviously, τ^+ and hence Δ need not be bounded when Θ is bounded.

The Θ-model of Definition 1, like synchronous models, abstracts away a number of intricate issues related to the actual sources of end-to-end delays. More specifically, in real systems, δ_{pq} consists not only of physical data transmission and processing times. Rather, delays due to the inevitable *scheduling* of the concurrent execution of multiple processes and message arrival interrupts/threads on every processor must be added to the picture. After all, every message that arrives over one of the n incoming links[5] must be processed (and/or put into memory for later processing) by a single CPU. Similarly, messages originating from multiple processes at the same processor must be scheduled for transmission over every outgoing link. Note that this remains true even if every pair of processes, along with its private link, is allocated to a dedicated pair of coprocessors: Unless every process pair existed in isolation from any other process pair, which is out of question in case of distributed algorithms, resource contention—and hence scheduling—is unavoidable.

Consequently, the end-to-end delay $\delta_{pq} = d_{pq} + \omega_{pq}$ between sender p and receiver q actually consists of a "fixed" part d_{pq} and a "variable" part ω_{pq}. The fixed part $d_{pq} > 0$ is solely determined by the processing speeds of p and q and the data transmission characteristics (distance, speed, etc.) of the interconnecting link. It determines the minimal conceivable δ_{pq} and does not usually vary much for different pairs p, q. The variable part $\omega_{pq} \geq 0$ captures all scheduling-related variations of the end-to-end delay, including

- precedences, resource sharing and contention, with or without resource preemption, which creates waiting queues,
- varying (application-induced) load,
- varying process execution times (which may depend on actual values of input parameters),
- occurrence of failures.

A *worst-case schedulability analysis* [27], [28], as well as a *best-case schedulability analysis* [29], [30], dealing with all the causes listed above must hence be conducted so as to establish the required upper and lower bounds $\overline{\tau}^+ \geq \tau^+$ and $\overline{\tau}^- \leq \tau^-$, respectively. Note that measurement-based approaches, which might suffice for non-critical systems, are out of question when safety-critical systems are considered.

For application-level end-to-end delays in moderately sized LAN-based systems, typical values are $\overline{\tau}_A^- = 100$ μs and $\overline{\tau}_A^+ = 10 \ldots 100$ ms. The resulting $\overline{\Theta}_A = 100 \ldots 1000$ would hence result in a prohibitively large detection time of our failure detector, cf. Theorem 2. This can be avoided by implementing a fast failure detector [4], however. Proper real-time scheduling algorithms—like head-of-the-line scheduling—can be applied at the FD-level here, which maintain a much smaller variability of ω_{pq} and hence a reasonably small $\overline{\Theta}$. Particular values for $\overline{\Theta}$, which obviously depends upon many system parameters, can of course only be determined by a detailed schedulability analysis; typical values reported for $\overline{\Theta}$ in real-time systems research [30], [31] are $2 \ldots 10$.

IV. Coverage Issues

Synchronous computational models assume that an upper bound $\overline{\tau}^+$ upon the end-to-end delays between correct processes is known a priori. Unfortunately, however, the worst-case schedulability analysis required for computing $\overline{\tau}^+$ is notoriously difficult. Almost inevitably, it rests on simplified models of reality (environments, technology) that may not always hold. As a consequence, $\overline{\tau}^+$ and hence any synchronous algorithm's correctness might be violated at run time in certain situations.

This is not necessarily the case with the Θ-model. First, its coverage is at least as high as the coverage of the corresponding synchronous model: Given that $\overline{\tau}^- > 0$ is determined by the best case scenario for the end-to-end delays and hence fairly easy to compute with very high

[5] In case of shared channel-based networks like Ethernet, a distributed message scheduling problem for resolving multiple access contention must be added to the picture. Since a shared channel makes it difficult for an adversary to increase τ^+ without simultaneously increasing τ^-, the arguments developed in Section IV apply a fortiori in this case.

coverage, $\Theta \leq \overline{\Theta} = \overline{\tau}^+/\overline{\tau}^-$ computed from $\overline{\tau}^+$ and $\overline{\tau}^-$ typically holds whenever $\tau^+ \leq \overline{\tau}^+$ holds. Second, if there is just *one* load/failure scenario where $\tau^+ > \overline{\tau}^+$ but still $\Theta \leq \overline{\Theta}$, its coverage is in fact higher.

This is likely to be the case, however, since τ^- typically also increases in situations where τ^+ increases. Note that τ^- must in fact only increase by α/Θ to compensate some $\tau^+ + \alpha$ here. To understand why, recall that τ^+, τ^- need only hold for *correct* processes that communicate with each other. If, say, $\delta_{xy} > \tau^+$ for only a limited number f of processes x and/or y, this can be attributed to process timing failures and is hence tolerated by proper fault-tolerant algorithms. An actual violation of τ^+ is hence a phenomenon that must occur in a significantly large part of the system, which makes it likely that it affects every processor to some extent.

For example, if $\delta_{pq} = \tau^+ > \overline{\tau}^+$ occurs at some processor pair p, q due to an increasing application-induced load in some significant portion of a system that employs "fully distributed" algorithms (as opposed to leader-based ones), additional messages are generated and sent to all processors in the system. Even if the computations of interest are the highest-priority ones and messages are transmitted using head-of-the-line policies, it may well be the case that this high-load situation does not allow the (often quite unlikely per se) best-case scenario $\delta_{rs} = \tau^-$ to occur at some processor pair s, r simultaneously: In fact, τ^- usually assumes that no blocking of high-priority processes by non-preemptable operations like sending a low-priority message occurs, which becomes increasingly difficult to maintain if the number of such messages increases. Consequently, the ratio of maximum over minimum delay need not grow as fast (if at all) as the maximum delay when the load increases.

In view of such correlations between minimum and maximum end-to-end delays, the probability of violating $\overline{\Theta}$ is (possibly significantly) smaller than the probability of violating $\overline{\tau}^+$. Hence, the coverage of $\overline{\Theta}$ is likely to be (significantly) larger than the coverage of $\overline{\tau}^+$; Θ-based designs like the failure detector of Section V should indeed surpass synchronous solutions in terms of coverage.

V. Principle of Operation

In this section, we introduce the principle of operation of our time-free perfect failure detector, which is based upon the consistent broadcasting primitive of [32]. Consistent broadcasting is implemented by means of two functions, *broadcast* and *accept*, which can be used to trigger a nearly simultaneous global event in the system: For example, a processor executing the clock synchronization algorithm of [32] (see Fig. 1) invokes *broadcast* to signal that it is ready for the resynchronization event to happen. It waits for the occurrence of this resynchronization event by calling *accept*, where it blocks until the event happens. The detailed semantics of consistent broadcasting is captured by three properties, namely, correctness, unforgeability and relay, which are defined as follows:

Definition 2 (Consistent Broadcasting Properties):

(C) *Correctness*: If at least $f+1$ correct processors call *broadcast* by time t, then *accept* is unblocked at every correct processor by time $t + \tau_{rt}^+$, for some $\tau_{rt}^+ \geq 0$.

(U) *Unforgeability*: If no correct processor calls *broadcast* by time t, then *accept* cannot unblock at any correct processor by $t + \tau_{rt}^-$ or earlier, for some $\tau_{rt}^- \geq 0$.

(R) *Relay*: If *accept* is unblocked at a correct processor at time t, then every correct processor does so by time $t + \tau_\Delta$, for some $\tau_\Delta \geq 0$.

Implementations of consistent broadcasting, providing specific values of τ_{rt}^+, τ_{rt}^- and τ_Δ, can be found in [26], [33].

Now consider a very simple algorithm, which executes infinitely many instances of consistent broadcasting $R = 0, 1, \ldots$ in direct[6] succession: If *broadcast*$_0$ is called by sufficiently many processors, *accept*$_0$ will eventually unblock and trigger *broadcast*$_1$, which in turn starts the same sequence for instance $1, \ldots$ Let σ_p^R denote the time when instance R terminates, that is, *accept*$_R$ unblocks and *broadcast*$_{R+1}$ starts at processor p. It is easy to show that the following holds for this algorithm:

Lemma 1 (Consecutive Consistent Broadcasting): Let $R \geq P \geq 0$ be two arbitrary instances of successive consecutive consistent broadcasting. Then,

$$|\sigma_p^R - \sigma_q^R| \leq \tau_\Delta \quad (1)$$

$$\sigma_p^R - \sigma_q^P \geq (R-P)\tau_{rt}^- - \tau_\Delta \quad (2)$$

$$\sigma_p^R - \sigma_q^P \leq (R-P)\tau_{rt}^+ + \tau_\Delta \quad (3)$$

for any two correct processors p, q.

Proof: Equation (1) follows immediately from the relay property (R), which establishes (2) and (3) for $P = R$ as well. We can hence assume that (2) holds for $k - 1 = R - P - 1 \geq 0$ and have to show that it holds

[6]This can of course be slowed down by introducing some delay in between, see [26].

for $k = R - P$ as well. Since processor p unblocks $accept_R$ at time σ_p^R, at least one non-faulty processor p_1 must have called $broadcast_R$ by time $\sigma_p^R - \tau_{rt}^-$ by the unforgeability property (U) of consistent broadcasting. Hence $\sigma_p^R - \sigma_{p_1}^{R-1} \geq \tau_{rt}^-$. By the induction hypothesis, we have $\sigma_{p_1}^{R-1} - \sigma_q^P \geq (R-P-1)\tau_{rt}^- - \tau_\Delta$, from which (2) follows immediately by adding up.

For the upper bound, we can assume that (3) holds for $k-1 = R-P-1 \geq 0$ and have to show that it holds for $k = R - P$ as well. Since we know that even the last correct processor unblocks $accept_{R-1}$ at some time $\sigma_{\max}^{R-1}$ satisfying $\sigma_{\max}^{R-1} - \sigma_q^P \leq (R-P-1)\tau_{rt}^+ + \tau_\Delta$ by the induction hypothesis, it follows from the correctness property (C) of consistent broadcasting that processor p must unblock $accept_R$ by time $\sigma_{\max}^{R-1} + \tau_{rt}^+$, from where (3) follows again by adding up. □

Equipped with this result, we can make the following important observation: Let us extend our simple successive consistent broadcasting algorithm by additionally broadcasting an $R+1$-*heartbeat* message (containing $R+1$ as data) when calling $broadcast_{R+1}$ at every processor. Now consider some correct processor p at time σ_p^R (when it sends its $R+1$-heartbeat): If p did not see a $P+1$-heartbeat, $P \leq R$, from some processor q (which should have been sent at time σ_q^P) by time σ_p^R, it must be the case that this heartbeat travels longer than $\sigma_p^R - \sigma_q^P \geq (R-P)\tau_{rt}^- - \tau_\Delta$ according to (2). Since we assumed that a heartbeat from a correct processor takes at most τ^+ to arrive, choosing

$$\Xi = R - P \geq \frac{\tau^+ + \tau_\Delta}{\tau_{rt}^-} \tag{4}$$

such that $\tau^+ \leq \Xi\tau_{rt}^- - \tau_\Delta$ ensures that p cannot miss q's $P+1$-heartbeat if q is correct. So if p did not see the $P+1$-heartbeat from q by time σ_p^R, q must have crashed and can safely be put on the list of suspects at processor p.

Of course, as it appears in (4), Ξ depends upon τ^+ and cannot be computed if this value is unknown. Our detailed analysis of consistent broadcasting in [26], [33] reveals, however, that actually $\tau_{rt}^- = 2\tau^-$ and $\tau_\Delta = 2\tau^+ - \tau^-$. Plugging this into our expression for Ξ yields $\Xi \geq 1.5\tau^+/\tau^- - 0.5$, i.e., a value that *only depends upon the ratio* $\Theta = \tau^+/\tau^-$. If an upper bound $\overline{\Theta}$ on Θ was known, we could compute Ξ even when bounds $\overline{\tau}^+$ and $\overline{\tau}^-$ on τ^+ and τ^-, respectively, are unknown! Note carefully that neither $\overline{\tau}^+$ nor $\overline{\tau}^-$ are built into our algorithm's code.

This result has important theoretical as well as practical implications. First, our FD algorithm (see Fig. 3) works also in a partially synchronous system where delay bounds are unknown, apparently contradicting [7]. Second, as discussed in Section IV, Θ may remain bounded even when τ^+ exceeds some assumed bound $\overline{\tau}^+$. In other words, there are conditions under which the correctness of our solution for implementing $\mathcal{P}$ depends neither on *load assumptions* nor on *failure rate assumptions*, apparently contradicting [8] as well.

The apparent contradictions are due to the fact that both [7] and [8] consider τ^- and τ^+ to be uncorrelated. For example, it can be assumed in [7] that the time t until correct suspicion in the strong completeness property (SC) is independent of τ^+, since the latter is unknown. This is not true if $\overline{\Theta}$ is known, however, since the FD could use any observed message delay as a bound for τ^- and then calculate a bound for τ^+ using $\overline{\Theta}$. Hence, an unknown upper bound upon the transmission delay alone is not sufficient to cause the impossibility result to apply. A similar argument applies for the impossibility of consensus in case of unbounded delays [8].

To further illustrate this important issue, we discuss an alternative implementation of $\mathcal{P}$, which works only in synchronous systems. The alternative algorithm is based upon the simple fact that consistent broadcasting allows to implement approximately synchronized clocks [32]. Each processor p must be equipped with a clock device $C_p(t)$ for this purpose, which is used to implement round R's clock $C_p^R(t) = C_p(t) + c_p^{R-1}$, by choosing a suitable time offset c_p^{R-1} set by the clock synchronization algorithm in round $R-1$. Note that we stipulate a round -1 clock $C_p^{-1}(t) = C_p(t)$ to be used in round 0. A single "composite" clock $C'_p(t)$ that equals the round clocks at the respective round switching times can be implemented atop of this, see [32, Sec. 7] for details.

Fig. 1 shows the simple clock synchronization algorithm of [32], which concurrently executes one instance of consistent broadcasting per resynchronization round R. Invoking $broadcast_R$ indicates that the calling processor is ready to resynchronize and hence start resynchronization round R, which is done every D seconds. Note carefully that the period D measures time according to every processor's clock, hence is a parameter of the algorithm that must assume some specific value. Consistent broadcasting ensures nearly simultaneous unblocking of $accept_R$—and hence resynchronization—of all non-faulty processors within τ_Δ by the relay property (R); a suitably chosen offset α is used to ensure that a clock is never set backwards.

```
Processor R ≥ 0:

    if C^{R-1}(t) = R · D /* ready to start C^R */
       → broadcast_R;
    fi
    if accept_R
       → C^R(t) := R · D + α; /* resynchronize */
    fi
```

Fig. 1. *A clock synchronization algorithm based upon consistent broadcasting*

The proof of correctness of this algorithm only depends upon the properties of consistent broadcasting. A detailed analysis in [32], [33] yields the following worst-case synchronization precision of the composite clock $C'_p(t)$:

$$\pi_{\max} = [(D-\alpha)(1+\rho) + \tau_{rt}^{+}]dr + \tau_\Delta(1+\rho) + \alpha, \quad (5)$$

where ρ is the worst-case clock drift and $dr = \rho(2+\rho)/(1+\rho)$.

In order to implement a failure detector, it suffices to add the same $R+1$-heartbeat broadcast as before when *broadcast*$_{R+1}$ is called. Obviously, processor q's $P+1$-heartbeat is sent at time $(P+1)D$ on q's clock, when processor p's clock reads at most $(P+1)D+\pi_{\max}$. When this heartbeat does not arrive by the time processor p performs *broadcast*$_{R+1}$, i.e., at time $(R+1)D$ on p's clock, q must have crashed if $\Xi = R-P$ is chosen such that $(R-P)D-\pi_{\max} \geq \tau^+$ (assuming clock drift $\rho = 0$ for simplicity). With $\alpha = 0$ and $D = \tau_{rt}^{-}$ as the minimal conceivable[7] choice of D, we find $\pi_{\max} = \tau_\Delta$ and hence

$$\Xi = R - P \geq \frac{\tau^+ + \tau_\Delta}{\tau_{rt}^{-}} \quad (6)$$

exactly as in (4). Consequently, both the algorithm of Fig. 3 and the one based upon Fig. 1 can (ideally) achieve approximately the same detection time. However, apart from the fact that the correctness of the above algorithm depends upon the correct operation of a processor's clock, it also needs the parameter D for properly adjusting the periodic *broadcasts*. In order to ensure that the computed precision $\pi_{\max}$ according to (5) is valid, D must be sufficiently large to ensure that any instance of consistent broadcasting is completed before the next one starts, which implies $D \geq 2\tau^+$. Therefore, the failure detector built atop of the algorithm in Fig. 1 is not time-free and may fail if τ^+ increases beyond the limit set forth by the choice of D.

[7]Those settings are smaller than the ones required by the analysis in [32], but serve as a means to illustrate the smallest conceivable lower bound on $R-P$ only.

VI. A Timer-Free Failure Detector

As explained in Section V, our failure detector implementation uses consistent broadcasting for implementing a time- and timer-free timeout mechanism based upon approximately synchronized rounds. Fig. 2 shows an implementation of the pivotal consecutive consistent broadcasting primitive, which works in presence of at most f arbitrary processor failures[8] if $n \geq 3f+1$. It has been derived from [32, Fig. 2] by simply replacing "*accept*" with "*broadcast*$_{R+1}$", which immediately starts the next instance.

```
Implementation of broadcast_R:
send (init, R) to all;

Code for process R:
cobegin
/* Concurrent block 1 */
if received (init, R) from f + 1 distinct processors
   → send (echo, R) to all [once]; /* sufficient evidence */
fi
/* Concurrent block 2 */
if received (echo, R) from f + 1 distinct processors
   → send (echo, R) to all [once]; /* sufficient evidence */
fi
if received (echo, R) from 2f + 1 distinct processors
   → broadcast_{R+1}; /* start next instance */
fi
coend
```

Fig. 2. *A simple consecutive consistent broadcasting algorithm*

Note carefully that all instances $R = 0, 1, \ldots$ of

[8]The classic perfect failure detector specification is of course only meaningful for simple crash failures. The existing work on muteness detector specifications [12], [16], [34]–[36] suggest to consider also more severe types of failures, however. Note also that there is a simpler implementation for f omission faulty processors, which only needs $n \geq 2f+1$ [32]. We use the more demanding version here, since it also tolerates arbitrary processor timing failures, i.e., f processors that inconsistently emit apparently correct messages at arbitrary times. An even more advanced hybrid version of consistent broadcasting, which tolerates hybrid processor and link failures, can be found in [26].

our algorithm must be created at boot time, in order to be ready when started by their predecessors. Again, σ_p^R denotes the time when round $R+1$ is started by *broadcast*$_{R+1}$.

By extending the proof of [32, Thm. 5], it is not difficult to show that the above algorithm satisfies the properties of consistent broadcasting in our system model:

Theorem 1 (Consistent Broadcasting Properties): In a system with $n \geq 3f+1$ processors, where at most f may be arbitrary faulty during the entire execution, the consistent broadcasting primitive of Fig. 2 created at boot time guarantees the properties of Definition 2 with $\tau_{rt}^+ = 2\tau^+$, $\tau_{rt}^- = 2\tau^-$, and $\tau_\Delta = \varepsilon + \tau^+ \leq \tau_{rt}^+$. System-wide, at most $2n$ broadcasts of $(1+\log_2 C_R)$-bit messages are performed by non-faulty processors, where C_R denotes the cardinality of the round number space (for R).

Proof: See [26] (or [33]). □

From Section V, we know that we only have to add the heartbeat-broadcast and the suspicion technique to the algorithm in Fig. 2 to get our timer-free failure detector as given in Fig. 3. Fortunately, we can simply use the (init, R) message as a processor's *R-heartbeat* message, since it is emitted at the time σ_p^{R-1} when *broadcast*$_R$ is started.

The following major Theorem 2 shows that the algorithm of Fig. 3 indeed implements a perfect failure detector in partially synchronous systems where only $\overline{\Theta} \geq \Theta = \tau^+/\tau^-$ is known.

```
  Global variables:
suspect[∀q] :=false; /* list of suspected processors */
saw_max[∀q] := 0; /* maximum heartbeat seen */

  Implementation start_R: /* Start heartbeat round R */
send (init, R) to all;

  Code process R:
cobegin
/* Concurrent block 1 */
if received (init, R) from q
   → if R > saw_max[q]
        → saw_max[q] := R; /* saw new heartbeat */
     fi
fi
if received (init, R) from f + 1 distinct processors
   → send (echo, R) to all [once]; /* sufficient evidence */
fi
/* Concurrent block 2 */
if received (echo, R) from f + 1 distinct processors
   → send (echo, R) to all [once]; /* sufficient evidence */
fi
if received (echo, R) from 2f + 1 distinct processors
   → ∀q: if R + 1 − Ξ > saw_max[q]
              → suspect[q] :=true; /* q has crashed */
         fi
      start_{R+1}; /* Start next heartbeat round */
fi
coend
```

Fig. 3. *Our time-free perfect failure detector algorithm*

Theorem 2 (Failure Detector Properties): Let Ξ be an integer with $\Xi \geq \lceil (3\overline{\Theta}-1)/2 \rceil$. In a system with $n \geq 3f+1$ processors, the algorithm given in Fig. 3 with all processes created at boot time implements a perfect failure detector with detection time at most $2(\Xi+2)\tau^+ - \tau^-$. During $C_R \geq 1$ rounds, every correct processor has running time within $[(2C_R+1)\tau^- - 2\tau^+, 2(C_R+1)\tau^+ - \tau^-]$ and broadcasts at most $2n$ messages of size at most $(1+\log_2 C_R)$ bits per round system-wide.

Proof: We have to show that the properties (SC) and (SA) of a perfect failure detector, given in Section II, are satisfied.

As far as strong accuracy (SA) is concerned, we use the argument developed in Section V. It confirms that every correct processor p gets the $P+1$-heartbeat of any alive processor q by the time σ_p^R when p emits its $R+1$-heartbeat, provided that Ξ is chosen according to (4). This evaluates to

$$\Xi \geq \frac{\tau^+ + \tau_\Delta}{\tau_{rt}^-} = \frac{3\tau^+ - \tau^-}{2\tau^-} = \frac{3\Theta - 1}{2} \qquad (7)$$

according to the latencies of consistent broadcasting given in Theorem 1. Inequality (7) obviously holds when Ξ is chosen as stated in our theorem.

As far as strong completeness is concerned, it is obvious that a processor q that has crashed stops emitting heartbeats. Hence, eventually, every alive processor recognizes that the expected heartbeat is missing and suspects q. The worst case for detection time occurs if a processor q crashes immediately after broadcasting its P-heartbeat, for some $P > 0$, at time σ_q^{P-1}. At time σ_p^R with $R - P = \Xi$, processor p recognizes that the $P+1$-heartbeat from q (which should have been emitted at time σ_q^P) is missing. According to (3) in Lemma 1, the detection time must hence be less than

$$\sigma_p^R - \sigma_q^{P-1} \leq (\Xi + 1)\tau_{rt}^+ + \tau_\Delta = 2(\Xi + 2)\tau^+ - \tau^-$$

according to Theorem 1 as asserted.

The bounds for the algorithm's running time for C_R rounds are determined by the running times of consistent broadcasting, hence follow immediately from (2) and (3) in conjunction with Theorem 1. The number of messages broadcast (= "send to all") and the message complexity follows immediately from Theorem 1. □

Remarks:

1) In the presence of crash (and even omission) failures, our failure detector guarantees that the properties (SC) and (SA) are maintained at all processors, i.e., even at faulty processors, until they crash. This is due to the fact that Theorem 1 is actually *uniform* [37], i.e., holds for such processors as well.
2) The algorithm does not need to know n, except for the size of the suspect list.
3) Initializing saw_max$[\forall q] := 0$ in Fig. 3 ensures that the very first 0-heartbeat is not used for suspecting a processor. This must be avoided, since we cannot assume that all correct processors start their execution within τ_Δ (which would be required to extend Lemma 1 to $R = -1$).
4) A theoretical disadvantage of our solution is that the message size is unbounded since it incorporates R. However, solutions to this problem—construction of a bounded round numbering scheme—are known, given that Ξ is known.

As a final remark, we note that the perfect failure detector algorithm of Fig. 3 could easily be transformed into an eventually perfect failure detector $\diamond\mathcal{P}$ that works also when Θ is unknown but finite, possibly after some unknown GST. All that needs to be done here is to increment Ξ every time a message from a suspected processor drops in later.

VII. Conclusions

In this paper, we showed how to implement a completely time- and timer-free perfect failure detector $\mathcal{P}$. The partially synchronous Θ-model utilized in our analysis requires a bound $\overline{\Theta}$ on the ratio $\Theta = \tau^+/\tau^-$ of maximum vs. minimum end-to-end delays only. Since there are systems or/and conditions where $\overline{\Theta} \geq \Theta$ holds true when some assumed upper bound $\overline{\tau}^+$ is violated (due to overloads, failure occurrences, etc.), our implementation of $\mathcal{P}$ achieves a coverage that is necessarily higher than what can be expected with synchronous solutions. Using the time immersion principle, our FD hence allows to build high-coverage distributed real-time systems based upon time-free (asynchronous) algorithms.

Part of our current/future work in this area is devoted to several extensions of our approach. In [26], we show how to arbitrarily reduce Ξ and system load by introducing additional delays, and how to deal with hybrid processor and link timing/value failures according to our perception-based failure model [25]. In [38], we present an alternative FD implementation based upon clock synchronization in the Θ-model [39], which also improves Ξ and allows us to get rid of the simultaneous booting assumption. We also found out that other time-free semantics, like the SDD problem [40] related to atomic commitment, can be solved in the Θ-model. Our ongoing research is primarily devoted to an experimental evaluation of our Θ-model, suitable real-time schedulability analysis techniques, and the extension to muteness detectors [12], [16], [35], [36].

References

[1] Flaviu Cristian and Christof Fetzer, "The timed asynchronous distributed system model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 642–657, 1999.

[2] Paulo Veríssimo, Antonio Casimiro, and Christof Fetzer, "The timely computing base: Timely actions in the presence of uncertain timeliness," in *Proceedings IEEE International Conference on Dependable Systems and Networks (DSN'01 / FTCS'30)*, 2000, pp. 533–542.

[3] Gèrard Le Lann, "On real-time and non real-time distributed computing," in *Proceedings 9th International Workshop on Distributed Algorithms (WDAG'95)*. September 13-15 1995, vol. 972 of *Lecture Notes in Computer Science*, pp. 51–70, Springer.

[4] J.-F. Hermant and Gerard Le Lann, "Fast asynchronous uniform consensus in real-time distributed systems," *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 931–944, Aug. 2002.

[5] Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty processor," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[6] Tushar Deepak Chandra and Sam Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.

[7] Mikel Larrea, Antonio Fernández, and Sergio Arévalo, "On the impossibility of implementing perpetual failure detectors in partially synchronous systems," in *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP'02)*, Jan. 2002.

[8] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer, "On the minimal synchronism needed for distributed consensus," *Journal of the ACM*, vol. 34, no. 1, pp. 77–97, Jan. 1987.

[9] Stephen Ponzio and Ray Strong, "Semisynchrony and real time," in *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92)*, November 1992, pp. 120–135.

[10] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer, "Bounds on the time to reach agreement in the presence of timing uncertainty," *Journal of the ACM (JACM)*, vol. 41, no. 1, pp. 122–152, 1994.

[11] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.

[12] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith, "Solving consensus in a byzantine environment using an unreliable fault detector," in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, Dec. 1997, pp. 61–75.

[13] Vijay K. Garg and J. Roger Mitchell, "Implementable failure detectors in asynchronous systems," in *Proceedings of the 18th Int. Conference on Foundations of Software Technology and Theoretical Computer Science (FST & TCS'98)*. 1998, LNCS 1530, pp. 158–169, Springer.

[14] Leslie Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, May 1998.

[15] Mikel Larrea, Antonio Fernández, and Sergio Arévalo, "Efficient algorithms to implement unreliable failure detectors in partially synchronous systems," in *Proceedings of the 13th International Symposium on Distributed Computing (DISC'99)*. Sept. 1999, LNCS 1693, pp. 34–48, Springer.

[16] Assia Doudou, Benoit Garbinato, Rachid Guerraoui, and André Schiper, "Muteness failure detectors: Specification and implementation," in *Proceedings 3rd European Dependable Computing Conference (EDCC-3)*, Prague, Czech Republic, Sept. 1999, LNCS 1667, pp. 71–87, Springer.

[17] Mikel Larrea, Antonio Fernández, and Sergio Arévalo, "Optimal implementation of the weakest failure detector for sloving consensus," in *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, 2000, p. 334.

[18] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera, "On the quality of service of failure detectors," in *Proceedings IEEE International Conference on Dependable Systems and Networks (ICDSN / FTCS'30)*, 2000.

[19] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg, "On quiescent reliable communication," *SIAM Journal of Computing*, vol. 29, no. 6, pp. 2040–2073, April 2000.

[20] Indranil Gupta, Tushar D. Chandra, and Germán S. Goldszmidt, "On scalable and efficient distributed failure detectors," in *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, Aug. 2001, pp. 170–179.

[21] Christof Fetzer, Michel Raynal, and Frederic Tronel, "An adaptive failure detection protocol," in *Pacific Rim International Symposium on Dependable Computing (PRDC 2001)*, Dec. 2001.

[22] Marin Bertier, Olivier Marin, and Pierre Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, Washington, DC, June 23–26, 2002, pp. 354–363.

[23] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg, "On implementing Omega with weak reliability and synchrony assumptions," in *Proceeding of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC'03)*, 2003, (to appear).

[24] Anchour Mostefaoui, Eric Mourgaya, and Michel Raynal, "Asynchronous implementation of failure detectors," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'03)*, San Francisco, CA, June 22–25, 2003, (to appear).

[25] Ulrich Schmid and Christof Fetzer, "Randomized asynchronous consensus with imperfect communications," in *22nd Symposium on Reliable Distributed Systems (SRDS'03)*, Oct. 6–8, 2003, (to appear).

[26] Gérard Le Lann and Ulrich Schmid, "How to maximize computing systems coverage," Tech. Rep. 183/1-128, Department of Automation, Technische Universität Wien, April 2003.

[27] Jane W. S. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[28] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgo C. Buttazzo, *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers, 1998.

[29] Ola Redell and Martin Sanfridson, "Exact best-case response time analysis of fixed priority scheduled tasks," in *Proceedings of the 14th Euromicro Workshop on Real-Time Systems*, June 2002, pp. 165–172.

[30] J.C. Palencia Gutiérrez, J.J. Gutiérrez Garcia, and M. Gonzáles Harbour, "Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems," in *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, June 1998, pp. 35–44.

[31] R. Ernst and W. Ye, "Embedded program timing analysis based on path clustering and architecture classification," in *Digest of Technical Papers of IEEE/ACM International Conference on Computer-Aided Design*. Apr. 1997, pp. 598–604, IEEE Computer Society.

[32] T. K. Srikanth and Sam Toueg, "Optimal clock synchronization," *Journal of the ACM*, vol. 34, no. 3, pp. 626–645, July 1987.

[33] Ulrich Schmid, "How to model link failures: A perception-based fault model," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Göteborg, Sweden, July 1–4, 2001, pp. 57–66.

[34] Danny Dolev, Roy Friedman, Idit Keidar, and Dahlia Malkhi, "Failure detectors in omission failure environments," in *Proc. 16th ACM Symposium on Principles of Distributed Computing*, Santa Barbara, California, 1997, p. 286.

[35] Dahlia Malkhi and Michael Reiter, "Unreliable intrusion detection in distributed computations," in *Proceedings of the 10th Computer Security Foundations Workshop (CSFW97)*, June 1997, pp. 116–124.

[36] Assia Doudou, Benoit Garbinato, and Rachid Guerraoui, "Encapsulating failure detection: From crash to byzantine failures," in *Reliable Software Technologies - Ada-Europe 2002*. June 2002, LNCS 2361, pp. 24–50, Springer.

[37] Vassos Hadzilacos and Sam Toueg, "Fault-tolerant broadcasts and related problems," in *Distributed Systems*, Sape Mullender, Ed., chapter 5, pp. 97–145. Addison-Wesley, 2nd edition, 1993.

[38] Josef Widder, Gérard Le Lann, and Ulrich Schmid, "A new timer-free perfect failure detector for partially synchronous systems," Tech. Rep., Technische Universität Wien, Institute of Computer Engineering, July 2003, (manuscript submitted).

[39] Josef Widder, "Booting clock synchronization in partially synchronous systems," in *Proceedings of the 17th International Symposium on Distributed Computing (DISC'03)*, Oct. 2003, (to appear).

[40] Bernadette Charron-Bost, Rachid Guerraoui, and André Schiper, "Synchronous system and perfect failure detector: Solvability and efficiency issues," in *Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks (DSN'00)*, New York, USA, 2000, pp. 523–532, IEEE Computer Society.