Weak Synchrony Models and Failure Detectors for Message Passing (k-)Set Agreement

Martin Biely, Peter Robinson, and Ulrich Schmid
Embedded Computing Systems Group (E182/2), Technische Universität Wien, Austria
{biely,robinson,s}@ecs.tuwien.ac.at

***Abstract*—The recent discovery of the weakest failure detector $\mathcal{L}$ for message passing set agreement has renewed the interest in exploring the border between solvable and unsolvable problems in message passing systems. This paper contributes to this research by introducing two novel system models $\mathcal{M}^{\text{anti}}$ and $\mathcal{M}^{\text{sink}}$ with very weak synchrony requirements, where $\mathcal{L}$ can be implemented. To the best of our knowledge, they are the first message passing models where set agreement is solvable but consensus is not. We also generalize $\mathcal{L}$ by a novel "$(n-k)$-loneliness" failure detector $\mathcal{L}(k)$, which allows to solve k-set agreement but not $(k-1)$-set agreement.We also present an algorithm that solves k-set agreement with $\mathcal{L}(k)$, which is anonymous in that it does not require unique process identifiers. This reveals that $\mathcal{L}$ is also the weakest failure detector for anonymous set agreement. Finally, we analyze the relationship between $\mathcal{L}(k)$ and other failure detectors, namely the limited scope failure detector $\mathcal{S}_{n-k+1}$ and the quorum failure detector Σ.**

***Index Terms*—Failure Detectors; Partial Synchronous Models; k-Set Agreement**

I. Introduction

In recent years, the quest for weak system models resp. failure detectors [1], which add just enough synchrony resp. failure information to purely asynchronous systems to circumvent impossibility results [2], has been an active research topic in distributed computing. Most work in this area falls into one of the following two categories: (1) Identifying weak failure detectors, and (2) strengthening the synchrony assumptions of the asynchronous model just enough to implement these weak failure detectors.

Due to the FLP impossibility result [2], which established that consensus among n processes with just $f = 1$ crash failures is impossible to solve in asynchronous systems, the focus of (1) was primarily the *consensus* problem. After the eventual leader oracle Ω [3], which eventually outputs the identifier of one correct process everywhere, was proved to be the weakest failure detector for solving consensus when a majority of the processes is correct, the research primarily shifted towards (2). The first implementation of Ω was provided by [4] and was based on rather strong synchrony assumptions (i.e., a variant of the partially synchronous model of [5]). The subsequent quest for the weakest synchrony assumptions for implementing Ω was started by [6], and resulted in a series of papers [6]–[9] in which the number of required timely links has been reduced considerably. In the most recent paper [9], it is shown that a single eventual moving f-source, i.e., a correct process that eventually has f (possibly changing) timely outgoing links in every broadcast, is sufficient for implementing Ω, and thus for solving consensus. Conversely, [10] revealed that Ω is sufficient for implementing an eventual $(n-1)$-source.

More recently, *set agreement* has been identified as a promising target for further exploring the solvability border in asynchronous systems. In [11], a failure detector called *anti*-Ω was shown to be the weakest for shared memory systems [12]. Like Ω, anti-Ω also returns the identifier of some process. The crucial difference to Ω is that anti-Ω eventually never outputs

A shorter version of this paper was accepted as brief announcement at DISC'09. Martin Biely and Peter Robinson have been supported by the Austrian BM:vit FIT-IT project *TRAFT* (proj. no. 812205) and the Austrian Science Foundation (FWF) project P20529, respectively. Correspondence to: Embedded Computing Systems Group (E182/2), Technische Universität Wien, Treitlstrasse 3, A-1040 Vienna (Austria). Fax: +43(1)58801-18297

the identifier of some *correct* process and does not need to stabilize on a single process identifier. A variant of anti-Ω, called anti-Ω_k [11], returns $n-k$ processes and was first conjectured to be the weakest failure detector [13] and later shown to be the weakest failure detector for k-set agreement [14], [15] in shared memory systems.

In [16], it has been shown that the quorum failure detector Σ is the weakest to implement shared memory in a message passing system when a majority of the processes may fail. Moreover, the combination of Σ and Ω was proved to be the weakest failure detector for solving consensus for any number of failures. For k-set agreement (with $k > 1$), however, an analogous combination (i.e., $\langle \Sigma, \text{anti-}\Omega_k \rangle$) is not the weakest failure detector, as k-set agreement is too weak for implementing atomic registers, whereas the proof of $\langle \Sigma, \Omega \rangle$ being the weakest failure detector for consensus critically depends upon the ability to implement atomic registers using consensus [17], [18]. Indeed, besides providing the weakest failure detector $\mathcal{L}$ for $(n-1)$-set agreement in message passing systems, [19] proved that this "loneliness" failure detector $\mathcal{L}$ is strictly weaker than Σ. Thus, the quest for message passing k-set agreement is still open.

This paper is devoted to k-set agreement in message passing systems, and provides the following contributions: (1) We introduce two novel system models $\mathcal{M}^{\text{anti}}$ and $\mathcal{M}^{\text{sink}}$, which provide just enough synchrony to implement $\mathcal{L}$ but are not strong enough to solve consensus. To the best of our knowledge, these models are the first message passing models where set agreement is solvable but consensus is not. (2) We define a novel failure detector $\mathcal{L}(k)$ that generalizes $\mathcal{L}$ to k-set agreement, and show that it is sufficient to solve k-set agreement. Since our $\mathcal{L}(k)$-based k-set agreement algorithm does not use process identifiers, it also works in anonymous systems. This implies that $\mathcal{L}$ is also the weakest failure detector for set agreement in anonymous systems. (3) We show that there is no algorithm that solves $(k-1)$-set agreement with $\mathcal{L}(k)$. (4) Finally, we compare $\mathcal{L}(k)$ to the limited scope failure detector $\mathcal{S}_{n-k+1}$ [20], which has also been employed for k-set agreement. For the "canonical" cases ($k = 1$ and $k = n-1$), we show that one of the two failure detectors is strictly stronger than the other; for any other choice of k, however, they are incomparable. As a consequence, neither $\mathcal{L}(k)$ nor $\mathcal{S}_{n-k+1}$ can be the weakest failure detector for general k-set agreement.

II. System Models and Problem Definition

The models we consider in this paper are based on the standard asynchronous model of [2], which we denote by $\mathcal{M}^{\text{async}}$ and introduce informally below. We consider a set Π of n distributed processes, which communicate via message passing over a fully-connected point-to-point network made-up of unidirectional links with finite but unbounded message delays. Links need not be FIFO but are assumed to be reliable[1] for simplicity. Every process executes an instance of a distributed algorithm and is modeled as a deterministic state machine. Its execution consists of a sequence of zero-time *steps*, where a single process performs a state transition according to its transition function, in addition to either receiving a (possibly empty) set of previously sent messages, or sending messages to an arbitrary set of processes (including itself). A *run* α of a distributed algorithm consists of a sequence of local steps of all the processes.

For analysis purposes, we assume the existence of a discrete global clock T, which ticks whenever a process takes a step. Note that processes do not have access to T.

Among the n processes, at most f can fail at any time by *crashing*. A process may crash within a step and does not take further steps afterwards. A *correct* process is one that never crashes. We call a process *alive at time* t if it does not crash before or at time t. Moreover, a process is alive in a time interval I when it is alive at every tick of T in I. The *failure pattern* of α is a function $F : T \to 2^{\Pi}$ that outputs the set of crashed processes for a given time t. Clearly, $\forall t \geqslant 0 : F(t) \subseteq F(t+1)$. Moreover, let $F = \bigcup_{t \geqslant 0} F(t)$ be the set of faulty processes. The set of possible failure patterns is called *environment*. In this paper we admit any environment that allows up to $n-1$ crashes.

A run α is *admissible in* $\mathcal{M}^{async}$ (1) if every correct process takes infinitely many steps, (2) a message is only received at time t by process p if it was sent

[1] In Section III-C, we discuss relaxations of this assumption.

by some process q to it at some time $t' \leqslant t$, and (3) every message sent to p is eventually received if p is correct. We say that an algorithms *halts* when it reaches a terminal state, where it remains for infinitely many steps.

A. k-Set Agreement

We now state the properties of the k-set agreement problem [21]. When $k = n - 1$, the problem is also referred to simply as set agreement. Every process starts with a proposal value v and must eventually irrevocably decide on some value as follows:

k-Agreement: Processes must decide on at most k different values.

Validity: If a correct process decides on v, then v was proposed by some process.

Termination: Every correct process must eventually decide.

Note that the agreement property we use ties together the decision values of all (correct or faulty) processes. For $k = 1$, it is hence equivalent to uniform consensus [22]. Moreover, note that it is well known that k-set agreement is impossible to solve in purely asynchronous systems when $f \geqslant k$ processes might crash [23]–[25].

B. Failure Detectors

A failure detector [1] $\mathcal{D}$ is an oracle that can be queried by processes in any step, before making a state transition. The behaviour of $\mathcal{D}$ in a run α depends on the failure pattern F, which defines the set of admissible *failure detector histories*. The value of a query of a process p in a step at time t is defined by the history function $H(p, t)$, which maps process identifiers and time to the *range* of output symbols of $\mathcal{D}$. Let A be an algorithm that uses $\mathcal{D}$ and let α be a run of A with failure pattern $F(t)$.

We denote the model where runs are admissible in $\mathcal{M}^{\text{async}}$ and processes can query failure detector $\mathcal{D}$ in any step as $(\mathcal{M}^{\text{async}}, \mathcal{D})$. If an algorithm A solves problem P in $(\mathcal{M}^{\text{async}}, \mathcal{D})$, we say that $\mathcal{D}$ *solves* P. We say that algorithm $A_{\mathcal{D}\to\mathcal{D}'}$ transforms $\mathcal{D}$ to $\mathcal{D}'$, if processes maintain output variables $output_{D'}$ that emulate failure detector histories of $\mathcal{D}'$ that are admissible for F. We say that $\mathcal{D}'$ is *weaker* than $\mathcal{D}$ and call $\mathcal{D}$ *stronger* than $\mathcal{D}'$, if such an algorithm $A_{\mathcal{D}\to\mathcal{D}'}$ exist. If there is also an algorithm $A_{\mathcal{D}'\to\mathcal{D}}$, we say that $\mathcal{D}$ and $\mathcal{D}'$ are *equivalent*. If no such algorithm $A_{\mathcal{D}'\to\mathcal{D}}$ exists, we say that $\mathcal{D}$ is *strictly stronger* than $\mathcal{D}'$; *strictly weaker* is defined analogously. If neither $A_{\mathcal{D}\to\mathcal{D}'}$ nor $A_{\mathcal{D}'\to\mathcal{D}}$ exists then we say that $\mathcal{D}$ and $\mathcal{D}'$ are *incomparable*. A failure detector $\mathcal{D}'$ is the *weakest for problem* P if $\mathcal{D}$ is weaker than any failure detector $\mathcal{D}$ that solves P.

Recently, it was shown in [19] that the "loneliness"-detector $\mathcal{L}$ is the weakest failure detector for message passing set agreement. Intuitively speaking, there is one (possibly crashed) process where $\mathcal{L}$ perpetually outputs FALSE, and, if all except one process p have crashed, $\mathcal{L}$ eventually outputs TRUE at p forever.

We now present our generalization of $\mathcal{L}$ for k-set agreement, which we denote by $\mathcal{L}(k)$ (with $\mathcal{L} = \mathcal{L}(n-1)$). Instead of loneliness it detects "$(n-k)$-loneliness", i.e., it detects the case where at most $n-k$ processes are still alive.

Definition 1: The *$(n-k)$-loneliness detector* $\mathcal{L}(k)$ outputs either TRUE or FALSE, such that for all environments $\mathcal{E}$ and $\forall F \in \mathcal{E}$ it holds that there is a set of processes $\Pi_0 \subseteq \Pi$, $|\Pi_0| = n - k$ and a correct process q such that:

$$\forall p \in \Pi_0\ \forall t\colon H(p,t) = \text{FALSE} \quad (1)$$

$$|F| \geqslant k \Longrightarrow \exists t\ \forall t' \geqslant t\colon H(q,t') = \text{TRUE} \quad (2)$$

Another class of failure detectors for k-set agreement are the limited scope failure detectors introduced by Mostefaoui and Raynal [20]. Such failure detectors have the strong completeness property (Eq. (4)) of the *strong* failure detector $\mathcal{S}$ [1], but their accuracy is limited to a set of processes called the scope (Eq. (3)). In the special case where the scope comprises all processes, $\mathcal{S}_n$ coincides with $\mathcal{S}$.[2] It was shown in [26] that $\mathcal{S}_{n-k+1}$ is sufficient for k-set agreement.

Definition 2: The *strong failure detector with x-limited scope* is denoted as $\mathcal{S}_x$ and is defined such that for all environments $\mathcal{E}$ and $\forall F \in \mathcal{E}$, there is a set

[2]For the case $k > f$ (which is not relevant here as $f = n - 1$) [20] also provides a transformation $T_{\mathcal{S}_k\to\mathcal{S}}$.

$Q \subseteq \Pi : \ |Q| = x$ such that:

$$\exists p \in (Q \setminus F) \ \forall t \ \forall q \in Q : \ p \notin H(q,t) \quad (3)$$

$$\forall p \in F \ \exists t \ \forall q \in \Pi : \ p \in H(q,t) \quad (4)$$

III. Weak System Models for Set Agreement

In this section, we introduce two system models $\mathcal{M}^{\text{anti}}$ and $\mathcal{M}^{\text{sink}}$ with very weak synchrony conditions. By implementing $\mathcal{L}$ in both of these models, we show that they are strong enough to solve set agreement. In order to allow this, we need to restrict the set of admissible runs of $\mathcal{M}^{\text{async}}$ by adding some—albeit very weak—synchrony conditions. While set agreement is solvable in either one of these models, the partial synchrony-like assumptions of $\mathcal{M}^{\text{sink}}$ are fundamentally different from the time-free message-ordering properties of model $\mathcal{M}^{\text{anti}}$.

A. The model $\mathcal{M}^{anti}$

In some applications, like VLSI Systems-on-Chip [27], a message-driven execution model [28]–[30], where computing steps are triggered by the arrival of messages instead of the passage of time, is advantageous over the usual time-driven execution model. The model $\mathcal{M}^{\text{anti}}$ presented in this section belongs to this category. Inspired by the round-trip-based model introduced in [31], [32], we specify our synchrony requirements as conditions on the order of round-trip message arrivals.

The basic round-trip protocol introduced in [31] proceeds in asynchronous rounds: Initially, every process p sends a $(query)$-message to all processes, including itself. If a process receives a $(query)$-message from some process q, it sends a $(resp)$-message to q. When p has received at least $n-f$ $(resp)$-message, it *starts a new round*, by sending out another $(query)$-message to all processes. Since we aim at $(n-1)$-set agreement with $f = n-1$ here, processes hence start their new round after receiving just 1 response. In the case where all other processes crash, the remaining process will end up receiving only messages sent by itself.

Definition 3 (Anti-Source): Let p be a correct or faulty process. Process p is an *anti-source*, if, whenever p sends a query to all processes, then the response from some other (possibly changing) process arrives at p before process p starts a new round.

Intuitively speaking, an anti-source is an (unknown) process whose round-trips with itself are never the fastest. Note that this definition also implies that the anti-source can never be the last remaining correct process.

Definition 4: Let α be a run of a distributed algorithm. Then, α is *admissible in $\mathcal{M}^{anti}$* if the following holds:

1) Run α is admissible in $\mathcal{M}^{\text{async}}$.
2) At least one process is an anti-source in α.

Algorithm 1 provides an implementation of the loneliness failure detector $\mathcal{L}$ in $\mathcal{M}^{\text{anti}}$: A process sets its $output_{\mathcal{L}}$ to TRUE if and only if it receives its own reply to its round-trip first. In every run, the anti-source p will always receive the reply message from some other process first and therefore never changes its variable $output_{\mathcal{L}}$ to TRUE.

Lemma 1: Let p be an anti-source in a run of Algorithm 1. Then, p never sets $output_{\mathcal{L}}$ to TRUE.

Proof: At the start of every round, process p sends a $(query)$-message to all other processes. By the definition of an anti-source, p always receives a $(resp)$-message to its query from some process $q \neq p$ as its first reply. Process p will therefore pass the test in Line 9 and set $alone \leftarrow$ FALSE. It follows that p will always pass the test in Line 12 and therefore $output_{\mathcal{L}}$ remains on FALSE forever. ■

Theorem 2: $\mathcal{L}$ is implementable in $\mathcal{M}^{\text{anti}}$.

Proof: Property (1) follows immediately from Lemma 1. We therefore only need to prove (2). Suppose that q is the only correct process in α. Then there is a time after which q does not receive any more messages from other processes. That is, there is a time t such that whenever q sends out a $(query)$-message, it only receives its own response, hence, it never sets $alone \leftarrow$ FALSE at any time $t' \geqslant t$. The one $(resp)$ message that q receives, however, is sufficient to subsequently cause q to set $output_{\mathcal{L}}$ to TRUE in Line 15. ■

B. The Model $\mathcal{M}^{sink}$

The model $\mathcal{M}^{\text{sink}}$ is similar to the weak-timely link (WTL) models [6]–[8], [33], which are derived from the classic partially synchronous models [5], [34].

Algorithm 1: $\mathcal{L}$ in Model $\mathcal{M}^{\text{anti}}$

```
1:  Vars:
2:  counter ∈ ℕ
3:  output_L, alone ∈ {TRUE, FALSE}
4:  Initially:
5:  output_L ← FALSE;
6:  startRound()

7:  upon receive (resp) from processes Q do
8:      counter ← |Q|
9:      if {p} ≠ Q then
10:         alone ← FALSE
11:     if counter ⩾ n − f then
12:         if alone = FALSE then
13:             startRound()
14:         else
15:             output_L ← TRUE

16: upon receive (query) from process q do
17:     send (resp) do q

18: procedure startRound()
19:     alone ← TRUE
20:     counter ← 0
21:     send (query) to all
```

Algorithm 2: $\mathcal{L}$ in Model $\mathcal{M}^{\text{sink}}$

```
1:  Vars:
2:  output_L ∈ {TRUE, FALSE}
3:  phase, maxSeen ∈ ℤ
4:  Initially:
5:  output_L ← FALSE
6:  phase ← −1
7:  maxSeen ← −1
8:  startPhase()

9:  every η steps do:
10: startPhase()

11: upon receive (alive, phase′) do
12:     maxSeen ← max(maxSeen, phase′)
13: upon expiration of timer do
14:     if maxSeen ⩾ phase then
15:         timer ← Φη + Δ
16:         start timer
17:     else
18:         output_L ← TRUE

19: procedure startPhase()
20:     phase ← phase + 1
21:     send (alive, phase) to all remote processes
```

Essentially, the WTL models assume that processes are partially synchronous [5] while trying to minimize the synchrony requirements on communication delays.

In the model $\mathcal{M}^{\text{anti}}$, there is no time bound on the duration of a round-trip as only the arrival order matters. Our second model $\mathcal{M}^{\text{sink}}$ enforces a similar order by means of explicit communication delay bounds and message timeouts, like the WTL models. A naïve approach would be to simply assume a bound on the round trip time, which is essentially equivalent to requiring a moving bi-directional link from one process. Note that this assumption would make one process permanently 1-*accessible* (in the notation of [8]), which turned out to be unnecessarily strong for our purposes.

As in [5], we assume two bounds Φ and Δ, where Φ bounds the relative speed of processes, whereas Δ bounds the transmission delay of a timely message m, measured in the number of steps of processes during the transmission of m. We also assume that processes can only either send messages or receive a possibly empty set of messages in a step. We say that a message m is delivered *timely* over the link (p, q) iff it is sent by p at time t and received by q not after the first reception step q takes at or after $t + \Delta$. Note that this definition implies that all messages sent to a crashed process (or a process that crashes before taking the decisive reception step) are considered to be delivered timely.

As in the WTL models (and in contrast to [5]), we do not assume Δ to hold for all messages. Rather, we base our synchrony conditions on a "sink", i.e., a process q that can always receive some messages timely.

Definition 5 (Sink): A process q is a *sink in a run* α if there is a correct process p such any message sent to q (before it may possibly crash) is delivered timely to q.

Note that we only consider p to be correct here to keep the definition simple. Indeed, when the sink q crashes, then p may crash as well, as long as it does so only after q. Note that this is actually the decisive difference between q being a sink and p being a perpetual 1-source (in the notation of [6]). This is not the end of the road, however, as this synchrony requirement can be further weakened when one con-

siders algorithms with a "round like" structure — that is, algorithms where each process repeatedly sends messages to all other processes, as it is often the case for heartbeat-based failure detector implementations. For such algorithms, one could also use the following Definition 6, where the timely process p may change. Note, however, that in contrast to the timely f-source model with moving timely links [33], we cannot rely on single (send-)event as a common reference point here.

Definition 6 (Sink'): A process q is a *sink in a run* α if, for every $i \geqslant 1$, there is a (possibly changing) process p such that the i-th message sent by p to q is delivered timely to q.

Note carefully that, since all messages sent to crashed processes are by definition delivered timely, a sink can also be a faulty process.

Definition 7 (Model $\mathcal{M}^{\text{sink}}$): Let α be a run of a distributed algorithm. Then, α is *admissible in* $\mathcal{M}^{sink}$ if the following holds:

1) Run α is admissible in $\mathcal{M}^{\text{async}}$.
2) There is a bound Φ, such that in every interval of Φ ticks on T every process that is alive throughout the interval takes at least one step.
3) At least one process is a sink in α.

At a first glance, it might be surprising that model $\mathcal{M}^{\text{sink}}$ is a non-eventual model, i.e., a model where all model properties must hold at all times. This is necessary in order to implement $\mathcal{L}$ (see Definition 1), which is a non-eventual failure detector. In fact, this is no peculiarity of set agreement: The weakest failure detector for $n-1$ resilient consensus is $\langle \Sigma, \Omega \rangle$, which is also non-eventual (see [35]).

Moreover, the definition of $\mathcal{L}$ makes it necessary that at least one process never falsely suspects "loneliness", i.e., the model parameters Φ and Δ must be known (and hold right from the start). While it would be sufficient if only the sink knew the real model parameters Φ and Δ, we do not assume that the sink is known in advance, so all processes must in fact know Φ and Δ. However, if the messages sent by some fixed process p to the sink q were always timely, it would be sufficient if just p and q respected Φ and Δ.

Algorithm 2 shows a simple protocol that implements $\mathcal{L}$ in model $\mathcal{M}^{\text{sink}}$: Variable $output_{\mathcal{L}}$ contains the simulated failure detector output. Every process p periodically sends out $(alive, phase)$-messages that carry the current phase-counter $phase$. In addition, it sets a timer that is implemented using simple step counting. If p does not receive a timely $(alive, phase')$-message that was sent by some other process in the current (or a future) phase, it sets $output_{\mathcal{L}} \leftarrow$ TRUE in Line 18. Note that the timer is not re-armed in this case; the algorithm continues to send $(alive, phase)$-messages to the other processes, however. It is important to observe that Algorithm 2 also works in *anonymous* systems, where processes do not have unique identifiers but can only distinguish their neighbors via local port numbers, cp. [36], [37]. In Section IV, we will also provide an anonymous algorithm that solves set agreement with $\mathcal{L}$.

The following lemma shows that the emulated $\mathcal{L}$ at a sink never outputs TRUE:

Lemma 3: If process q is a sink, then q never executes Line 18 of Algorithm 2.

Proof: We must show that q receives the $(alive, k)$-message from some process before its timer runs out the $(k+1)$-st time, for any $k \geqslant 0$. Since q is a sink, the $(alive, k)$ is delivered timely to q from some process p. Let $T(\psi)$ denote the time on our global clock T when event ψ takes place somewhere in the system. Suppose that p sends the $(alive, k)$-message in some step ψ_p. By the code of the algorithm, process p must have executed $k\eta$ steps.[3] Since processes are partially synchronous, we have $T(\psi_p) \leqslant \Phi k\eta$. Now suppose that q's timer expires in step ψ_q for the $(k+1)$-st time. That is, q has made $(k+1)(\Phi\eta + \Delta)$ steps by ψ_q. Obviously, we have $T(\psi_q) \geqslant (k+1)(\Phi\eta + \Delta)$. Considering that the message from p to q is delivered timely, we are done if we can show $T(\psi_p) + \Delta \leqslant T(\psi_q)$. We find $\forall k \geqslant 0\colon\ T(\psi_p) + \Delta \leqslant k\Phi\eta + \Delta < (k+1)\Phi\eta + \Delta \leqslant (k+1)(\Phi\eta + \Delta) \leqslant T(\psi_q)$, which completes the proof. ■

Theorem 4: Algorithm 2 implements failure detector $\mathcal{L}$ in model $\mathcal{M}^{\text{sink}}$ for $f = n - 1$.

Proof: Let α be a run of Algorithm 2 in $\mathcal{M}^{\text{sink}}$, and p be any sink. Lemma 3 implies that p perpetually outputs FALSE in α (until it crashes), so (1) holds.

For proving (2), suppose that $n-1$ processes crash in α. Since there must be some process from which p

[3]For simplicity, we assume that all processes initially start up at the same time.

receives timely messages, p cannot be the only correct process in α. Let $q \neq p$ be the only correct process in α. Since q only sends its $alive$-messages to remote processes and no other process is alive, q's timer will eventually expire without receiving any message, i.e., q will set $output_{\mathcal{L}} \leftarrow$ TRUE in Line 18. ■

C. Comparing $\mathcal{M}^{sink}$ to an f-Source Model

It is interesting to compare $\mathcal{M}^{\text{sink}}$ to the f-source model $\mathcal{S}^{\rightarrow}_{f*}$ of [33], which is strong enough to solve consensus by implementing Ω for $f < n/2$. Just like $\mathcal{M}^{\text{sink}}$, model $\mathcal{S}^{\rightarrow}_{f*}$ assumes that processes are partially synchronous and that processes can send a message to multiple receivers in a single step. Moreover, in every run that is admissible in $\mathcal{S}^{\rightarrow}_{f*}$, there is some correct process p that is an eventual moving-f-source, i.e., p has at least f outgoing timely links, i.e., messages are delivered timely, to a (possibly changing) set of f processes.

Since we consider a perpetual model $\mathcal{M}^{\text{sink}}$, with failure patterns where up to $n-1$ processes can crash, we will compare it to a perpetual model $\mathcal{S}^{\rightarrow}_{n-1*}$ that contains at least one *perpetual* moving-$(n-1)$-source p. Clearly, since $n-1$ are all remote processes, there is no point in assuming that these links are moving here. Since every process $q \neq p$ receives all messages from p timely, every such q is a sink. Hence, it is not difficult to show that $\mathcal{M}^{\text{sink}}$ has weaker synchrony requirements than $\mathcal{S}^{\rightarrow}_{n-1*}$:

Theorem 5: Any run α that is admissible in the (perpetual) model $\mathcal{S}^{\rightarrow}_{n-1*}$ is admissible in $\mathcal{M}^{\text{sink}}$, but there are runs admissible in $\mathcal{M}^{\text{sink}}$ that are not admissible in $\mathcal{S}^{\rightarrow}_{n-1*}$.

Proof-Sketch: The first part of the theorem follows directly by the previous discussion. To see that there are runs that are only admissible in $\mathcal{M}^{\text{sink}}$ but not in $\mathcal{S}^{\rightarrow}_{n-1*}$, consider for example the run α where the sink is initially dead. As there are no other synchrony requirements on any remaining messages in α, the existence of a timely $(n-1)$-source is not guaranteed. ■

In $\mathcal{S}^{\rightarrow}_{f*}$, links are assumed to be reliable as in $\mathcal{M}^{\text{sink}}$, but it is argued in [33] that this assumption is unnecessary. In some other work [6], [8] on weak system models for implementing Ω, links can be unreliable. This leads us to the question of whether we could drop the reliable links assumption for our models as well. We can answer this question in the affirmative: If we are only interested in implementing $\mathcal{L}$ in $\mathcal{M}^{\text{sink}}$, it suffices that all messages sent over a timely link arrive; all other links may be totally unreliable. If we also wanted to solve (non-uniform) set-agreement on top of $\mathcal{L}$, however, links would need to be at least fair lossy.[4] Note that there is a problem when considering our definition of set-agreement as it requires uniform agreement: For uniform problems, [38] shows that reliable links are necessary when more than half of the processes might crash, which is the interesting case for set agreement. In contrast, when implementing Ω and consensus atop of Ω, $n > 2f$ is required anyway, and therefore one can use the reliable link simulation of [38], which works for $n > 2f$ and fair lossy links. However, when reverting to non-uniform (correct-restricted/failure-insensitive [39], [40]) set agreement, i.e., set agreement that requires the set agreement property to hold for *correct* processes only, we could assume fair lossy links as well.

D. Consensus Impossibility

Given these similarities (cf. Theorem 5) with a model where consensus is solvable, the question of whether our models are also strong enough to solve consensus arises naturally. We now show that this question can be answered in the negative. Due to the fact that our models are very close to the asynchronous model, the proof is surprisingly simple.

Theorem 6: Consider a message passing system of size $n \geqslant 3$, where up to $n-1$ processes may crash. There is no algorithm that solves consensus in model $\mathcal{M}^{\text{anti}}$ or in model $\mathcal{M}^{\text{sink}}$.

Proof: Suppose, for a contradiction, that there is an algorithm A^{sink} (resp. A^{anti}) that solves consensus in model $\mathcal{M}^{\text{sink}}$ (resp. $\mathcal{M}^{\text{anti}}$).

$\mathcal{M}^{\text{sink}}$: Consider a run α of A^{sink} where some process p is initially dead. Since p satisfies the definition of a sink, there are no other synchrony requirements on the links connecting the remaining correct processes.

[4] Alternatively, using our Algorithm 3 it suffices when one link is (perpetually) reliable and timely, and all others drop all messages. By abandoning failure detectors and merging Algorithms 2 and 3, the exact number of messages for which the link has to be timely could be determined.

Algorithm 3: Solving k-set agreement with $\mathcal{L}(k)$

```
1:  x ← v
2:  rnd ← 0
3:  initially send (round, 0, x) to remote processes

4:  in any later step:
5:  if L(k) = TRUE then
6:      send (dec, x) to all
7:      decide x
8:      halt
9:  if received (dec, y) then
10:     send (dec, y) to all
11:     decide y
12:     halt
13: if received (round, rnd, y) from n−k remote processes then
14:     S ← {y_1, ..., y_{n−k}} ∪ {x}
15:     x ← min(S)
16:     if rnd = k + 1 then
17:         send (dec, x) to all
18:         decide x
19:         halt
20:     rnd ← rnd + 1
21:     send (round, r, x) to all remote processes
```

Hence the set of the runs where p is initially dead is indistinguishable from the set of runs generated by A^{sink} in a system $\mathcal{M}^{\text{async}}$ with just $n-1 \geqslant 2$ processes, where processes are partially synchronous, all links are asynchronous, and $f = n - 2 \geqslant 1$ processes can still crash. This contradicts the impossibility results in [34, Table 1].

$\mathcal{M}^{\text{anti}}$: Consider a run α of A^{anti}, where some process p is initially dead. Since p satisfies the definition of an anti-source, there are no other synchrony requirements at all in $\mathcal{M}^{\text{anti}}$. Therefore, the set of these runs where p is initially dead is indistinguishable from the set of runs generated by A^{anti} in a system $\mathcal{M}^{\text{async}}$ with $n - 1 \geqslant 2$ processes, where still $f = n - 2 \geqslant 1$ processes can crash. This, however, contradicts the FLP impossibility [2]. ∎

IV. Solving k-Set Agreement with $\mathcal{L}(k)$

In this section, we present an algorithm that solves k-set agreement with the $\mathcal{L}(k)$ failure detector introduced in Definition 1. The original algorithm for solving $(n-1)$-set agreement with $\mathcal{L}$ [19] requires a total order on process identifiers. Algorithm 3, in contrast, also works in anonymous systems.

We denote by X^r the possibly empty array containing all x-values after the assignment in line 15 while the round variable rnd was set to r. We assume that X^r is ordered by decreasing values, i.e., $X^r[1]$ is the maximal value, if it exists. Furthermore, we denote the number of nonempty entries in X^r by $|X^r|$.

Lemma 7: For any round $r \geqslant 0$, the number of unique values in X^r is $u_r \leqslant k - a_r$, where a_r is the number of processes which never sent $(round, r, x)$.

Proof: First, we observe that x is updated by a process p only after receiving $n - k$ $(round, r, y)$ messages from other processes.

Let p be the process which assigns the largest value in line 15. Since any process p sets x to the minimum of the $n - k$ round r values received, there must be $n - k - 1$ messages containing values $y \geqslant x$ among those received by p.

Considering that $|X^r| \leqslant n - a_r$, it follows that only $n - a_r - (n - k + 1) \leqslant k - a_r - 1$ values in X^r can be smaller than p's minimum. Thus, processes assign at most $k - a_r$ different values to x and subsequently send them as $(round, r + 1, x)$-messages. ∎

Lemma 8: Processes do not decide on more than k different values.

Proof: Regarding the number of different decision values, processes deciding due to receiving a (dec, y) message (line 11) make no difference, since some other process must have decided on y using another method before. Thus we can ignore this case here.

What remains are decisions due to $\mathcal{L}(k)$ being TRUE (line 7) and due to having received $n - k$ messages in round $k + 1$ (line 18). For each $r \geqslant 0$, we denote by ℓ_r the number of processes which have decided due to their failure detector output being TRUE while their $rnd = r$. Thus the number of processes that have decided in line 7 with $rnd \leqslant r$ for some $r \geqslant 0$ is $\Sigma_{s=0}^{r} \ell_s$. In the following we use Σ^r as an abbreviation for this sum. Since processes halt after deciding, we can deduce that the number of processes which do not send round r messages a_r, is at least Σ^{r-1}. Thus, Lemma 7 tells us that $u_r \leqslant k - \Sigma^{r-1}$.

Now we assume by contradiction, that there are actually $D > k$ decisions, with $D = u_{k+1} + \Sigma^{k+1}$, that is the number of different values decided on in line 18 plus those that decided based on $\mathcal{L}(k)$. Thus we get $u_{k+1} > k - \Sigma^{k+1}$, and by using the above property of

u_r, we deduce that $\Sigma^{k+1} > \Sigma^k$, and thus $\ell_{k+1} \geqslant 1$. These processes must have decided on some values in X^k, however, which leads to the realisation that $D = u_k + \Sigma^k$. We can repeat this argument until we reach $D = u_1 + \Sigma^0 = u_1 + \ell_0$. Here, Lemma 7 gives us the trivial upper bound $u_1 \leqslant k$, which entails the requirement $\ell_0 \geqslant 1$ as $D > k$.

By now, we have shown that, assuming $D > k$ decisions $\ell_r \geqslant 1$ is required for $r \in \{0, \ldots, k+1\}$. In other words we have deduced that $\Sigma^{k+1} \geqslant k + 1$ processes have decided due to their $\mathcal{L}(k)$ output being TRUE. This contradicts property (2) of $\mathcal{L}(k)$, thus proofing the Lemma. ■

Theorem 9: Algorithm 3 solves k-set agreement in the anonymous asynchronous system augmented with $\mathcal{L}(k)$.

Proof: Validity is evident, since no value other than the initial values v of processes are ever assigned directly or indirectly to x. *k-Agreement* follows from Lemma 8, and since either $n - k$ processes send messages in each round or some process has $\mathcal{L}(k) =$ TRUE, every correct process *terminates*. ■

From [19], we know that $\mathcal{L}$ can be extracted anonymously from any failure detector D which solves set-agreement using some algorithm A: Every process executes an independent instance of A using D as failure detector. The simulated $\mathcal{L}$ outputs TRUE at p only after A has terminated at p. In conjunction with Theorem 9, this implies the following fact:

Corollary 10: $\mathcal{L}$ is the weakest failure detector for set agreement in anonymous message passing systems.

Theorem 9 showed that $\mathcal{L}(k)$ is sufficient for k-set agreement. We now prove that it is not (much) stronger than necessary, as $\mathcal{L}(k)$ is too weak to solve $(k-1)$-set agreement.

Theorem 11: No algorithm can solve $(k-1)$-set agreement with $\mathcal{L}(k)$, for any $2 \leqslant k \leqslant n-1$.

Proof: We assume by contradiction that such an algorithm A exists. Now consider the failure detector history where processes $p_1, \ldots, p_k$ output TRUE perpetually, while the other processes output FALSE. Clearly, this defines a legal history for $\mathcal{L}(k)$ in a run where the $n-k$ processes $p_{k+1}, \ldots, p_n$ crash initially. For the remaining k processes, the failure detector provides no (further) information about failures, as it outputs TRUE perpetually. Since A is able to solve $(k-1)$-set agreement in any such run by assumption, it can also be used to solve $(k-1)$-set agreement in an asynchronous system of k processes, equipped with a *dummy* failure detector [41] that always outputs TRUE. Clearly, this contradicts the $(n-1)$-set agreement impossibility in a system of n processes [24], [25], [42]. ■

V. Relation of $\mathcal{L}(k)$ to $\mathcal{S}_{n-k+1}$ and Σ

In this section, we discuss how the $\mathcal{L}(k)$ failure detector relates to the limited accuracy failure detector S_{n-k+1} (see Definition 2). Theorem 15 shows that, except in the canonical cases $k = 1$ and $k = n-1$, these failure detectors are incomparable. The proof consists of the following series of technical lemmas:

Lemma 12: $\mathcal{L}(1)$ is stronger than $\mathcal{S}_n = \mathcal{S}$.

Proof: In order to show that $\mathcal{L}(1)$ is stronger than $\mathcal{S}_n$, we show that we can implement $\mathcal{S}_n$ with $\mathcal{L}(1)$. For $\mathcal{S}$, we have to find one correct process which is never suspected by anyone (weak accuracy), while eventually every faulty process is suspected (strong completeness). As $\mathcal{L}(1)$ must output TRUE at one correct process only if at least one process has crashed, the idea of the transformation is quite simple: A process always outputs the empty set as its suspicion list, unless (1) it is instructed otherwise by another process, or (2) its $\mathcal{L}(1)$ outputs TRUE. Since $n-1$ processes must output FALSE, case (2) can only occur at a single process p, which then sends a message to all other processes telling them to suspect everyone but p.

Strong completeness follows, because if one (or more) processes crash, $\mathcal{L}(1)$ will eventually output TRUE at some correct p, causing all faulty processes to be suspected (along with all correct processes apart from p) by all other processes. Weak accuracy follows from p never being suspected. ■

Next we consider the general case, i.e., $k > 1$. Here we show that, except in case $k = 1$, no $\mathcal{L}(k)$ is stronger than even the weakest (non-trivial) instance of $\mathcal{S}_x$, namely $\mathcal{S}_2$.

Lemma 13: $\mathcal{L}(k)$ is not stronger than $\mathcal{S}_2$, for all $k > 1$.

Proof: Assume a transformation T exists, which implements $\mathcal{S}_2$ based on $\mathcal{L}(k)$. Now consider a run

α where only p crashes initially, i.e., $|F| = 1$. Since $|F| \leqslant 1 < k$, $\mathcal{L}(k)$ can perpetually output FALSE at all processes in α. By the *strong completeness* property of $\mathcal{S}_2$, transformation T must ensure that there is some time t such that all processes suspect p.

Now consider a different run α' where all messages from p to other processes are delayed until $t' > t$. Moreover, assume that all processes except p crash at some time $t'' > t'$ in α' and consider the failure detector history H where $\mathcal{L}(k)$ outputs TRUE at p, and FALSE at all processes $q \neq p$. Clearly H is a valid history for α' and p has to be in the output suspect-list of all processes $q \neq p$ by time t, as α' is indistinguishable until t' from α. But since in α' all processes except p crash at time t'', p is the only correct process, but was suspected by all other processes, contradicting 2-weak accuracy (3). ■

Lemma 14: $\mathcal{S}_{n-k+1}$ is not stronger than $\mathcal{L}(k)$, for $k < n-1$.

Proof: We again assume by contradiction that a suitable transformation algorithm T exists, which builds $\mathcal{L}(k)$ from $\mathcal{S}_{n-k+1}$. Let α_i be a run where all processes except p_i crash initially. Then, as p_i is the only remaining correct process, it must eventually set $output_{\mathcal{L}(k)}$ to TRUE at some time t_i. By applying this construction to a set $S = \{p_1, \ldots, p_{k-1}\}$ of processes, we get a time $t = \max(t_1, \ldots, t_{k-1})$, when every p_i has set $output_{\mathcal{L}(k)}$ to TRUE in the respective α_i.

Now consider a run α where every process in a set $S = \{p_1, \ldots, p_{k-1}\}$ suspects every other process, that is $\forall p_i \in S\colon H(p_i, t) = \Pi \setminus \{p_i\}$, and process p_k never suspects anybody. Moreover, the delivery of all messages from other processes to any process in S is delayed until time t.

Then, for any process $p_i \in S$, the run α is indistinguishable from the run α_i where only p_i is alive and so all $k-1$ processes in S have set $output_{\mathcal{L}(k)}$ to TRUE by time t in α. Note that this does not violate the $(n-k+1)$-*weak accuracy* of $\mathcal{S}_{n-k+1}$. Now suppose that all processes in S crash at some time $t' > t$, and also assume that p_k initially crashes.

Since k processes crash in α, it follows by the fact that T implements $\mathcal{L}(k)$ that at least one of the remaining processes $p_{k+1}, p_{k+2}, \ldots, p_n$ has to set $output_{\mathcal{L}(k)}$ to TRUE eventually; w.l.o.g. let p_{k+1} be that process and t_{k+1} be the time when it does so. Since $n \geqslant k+2$, we can assume that p_{k+1} crashes after t_{k+1} as well, and repeat the argument for process p_{k+2}. But now, $k+1$ processes have set their output variable $output_{\mathcal{L}(k)}$ to TRUE, which contradicts the requirement (2) of $\mathcal{L}(k)$. ■

From Lemma 12 and 14 it follows that $\mathcal{S}_n$ is strictly weaker than $\mathcal{L}(1)$. Moreover, Lemma 13 and the result that $\mathcal{L} = \mathcal{L}(n-1)$ is the weakest failure detector for set agreement [19] implies that $\mathcal{S}_2$ is strictly stronger than $\mathcal{L}(n-1)$. For the remaining choices of k, we get that $\mathcal{S}_{n-k+1}$ and $\mathcal{L}(k)$ are not comparable by Lemma 14 and 13, which completes the proof of Theorem 15.

Theorem 15: Failure detector $\mathcal{S}_n$ is strictly weaker than $\mathcal{L}(1)$, and $\mathcal{S}_2$ is strictly stronger than failure detector $\mathcal{L}(n-1)$. For $1 < k < n-1$, $\mathcal{L}(1)$ and S_{n-k+1} are incomparable.

Recalling the result of [43], this immediately implies:

Corollary 16: Neither $\mathcal{L}(k)$ nor $\mathcal{S}_{n-k+1}$ is the weakest failure detector for general message passing k-set agreement.

Despite Corollary 16, however, $\mathcal{L}(k)$ appears to be a promising candidate for the weakest failure detector for message passing k-set agreement in anonymous systems, i.e., without unique process ids.

As a final relation, we now explore the relation between $\mathcal{L}(k)$ and Σ. The general case for $\mathcal{L}(k)$ can be deduced from the more specific result of [19, Lemma 4] by finding and investigating a suitable partitioning.

Lemma 17: $\mathcal{L}(k)$ is not stronger than Σ, if $n > 2$ and $k \geqslant 2$.

Proof: Assume that there exists an algorithm A that transforms $\mathcal{L}$ into Σ. Consider the partitioning of Π given by $P = \{\{p_1\}, \{p_2\}, P_3\}$. Since $n-1 \geqslant k \geqslant 2$ this is a valid partitioning. Moreover, assume two runs r_1 and r_2 such that process p_i is correct in run r_i and all other processes are faulty from the beginning. Moreover, the output of $\mathcal{L}(k)$ at process p_i is TRUE from the beginning as well. Since A must guarantee completeness for Σ, it eventually has to output $\{p_i\}$ in run r_i say at time t_i. Now imagine a run r in which the processes p_1 and p_2 are correct and the output of $\mathcal{L}$ is TRUE from the beginning. Additionally, no message of a process from a different partition is received by these two processes before time $t = \max\{t_1, t_2\}$. Then, runs r_1 and r_2 are indistinguishable from run r before

time t. Therefore, the output of Σ at p_i at time t_i will be the same as in r_i. But this contradicts the intersection property of Σ. So there exists no such algorithm A. ∎

Moreover, Theorem 17 implies that either $\mathcal{L}(k)$ is strictly weaker than Σ or the two are not comparable. In either case, when adding another failure detector to Σ, say anti-Ω_k, the combined failure detector is stronger than Σ, so the same applies to it. Consequently:

Corollary 18: $\langle \Sigma, \text{anti-}\Omega_k \rangle$ is not the weakest failure detector for solving k-set agreement in message passing systems.

VI. Conclusions

We introduced two novel message passing models that provide just enough synchrony for set agreement but not enough for consensus. We also showed how to implement the weakest loneliness failure detector $\mathcal{L}$ for set agreement in these models, and proved that $\mathcal{L}$ is also the weakest failure detector for set agreement in anonymous systems. Finally, we generalized $\mathcal{L}$ to the $(n-k)$-loneliness failure detector $\mathcal{L}(k)$, which allows to solve k-set agreement but not $(k-1)$-set agreement. Part of our future research will be devoted to the relationship between (anonymous and non-anonymous) failure detectors and the synchrony conditions necessary for implementing them. One direction is the question of whether our models can be generalized for k-set agreement. Tightly connected to this question is the still ongoing search for the weakest failure detector for message passing k-set agreement.

References

[1] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

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

[3] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, June 1996.

[4] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 52–59, Nürnberg, Germany, October 2000.

[5] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[6] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing Omega with weak reliability and synchrony assumptions. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*, pages 306–314, Boston, Massachusetts, USA, July 2003.

[7] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *Proceedings of the 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 328–337, St. John's, Newfoundland, Canada, 2004. ACM Press.

[8] Dahlia Malkhi, Florin Oprea, and Lidong Zhou. Ω meets paxos: Leader election and stability without eventual timely links. In *Proceedings of the 19th Symposium on Distributed Computing (DISC'05)*, volume 3724 of *LNCS*, pages 199–213, Cracow, Poland, 2005. Springer Verlag.

[9] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Brief announcement: Chasing the weakest system model for implementing omega and consensus. In *Proceedings Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (formerly Symposium on Self-stabilizing Systems) (SSS 2006)*, LNCS, pages 576–577, Dallas, USA, Nov. 2006. Springer Verlag.

[10] Martin Biely, Martin Hutle, Lucia Draque Penso, and Josef Widder. Relating stabilizing timing assumptions to stabilizing failure detectors regarding solvability and efficiency. In *9th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume 4838 of *Lecture Notes in Computer Science*, pages 4–20, Paris, November 2007. Springer Verlag.

[11] Piotr Zielinski. Anti-Ω: the weakest failure detector for set agreement. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 55–64, New York, NY, USA, 2008. ACM.

[12] Piotr Zielinski. Automatic classification of eventual failure detectors. In *Distributed Computing, 21st International Symposium, DISC 2007*, volume 4731 of *Lecture Notes in Computer Science*, pages 465–479, Lemesos, Cyprus, September 2007. Springer Verlag.

[13] Michel Raynal. k-anti-Ω, August 2007. Rump session at PODC 2007.

[14] Eli Gafni and Petr Kuznetsov. The weakest failure detector for solving k-set agreement. Technical report, 2009-06, TU Berlin, February 2009.

[15] Eli Gafni and Petr Kuznetsov. The weakest failure detector for solving k-set agreement. In *28th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2009)*, 2009. (to appear).

[16] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 338–346. ACM Press, 2004.

[17] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[18] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[19] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The weakest failure detector for message passing set-agreement. In *DISC '08: Proceedings of the 22nd international symposium on Distributed Computing*, pages 109–120, Berlin, Heidelberg, 2008. Springer-Verlag.

[20] Achour Mostéfaoui and Michel Raynal. Unreliable failure detectors with limited scope accuracy and an application to consensus. In *FSTTCS*, pages 329–340, 1999.

[21] Soma Chaudhuri. More choices allow more faults: set consensus problems in totally asynchronous systems. *Inf. Comput.*, 105(1):132–158, 1993.

[22] Bernadette Charron-Bost and André Schiper. Uniform consensus is harder than consensus. *J. Algorithms*, 51(1):15–37, 2004. Also published as Tech. Rep. DSC/2000/028, Ecole Polytechnique Fédérale de Lausanne.

[23] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100, New York, NY, USA, 1993. ACM.

[24] Maurice Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 111–120, New York, NY, USA, 1993. ACM.

[25] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.

[26] Achour Mostéfaoui and Michel Raynal. k-set agreement with limited accuracy failure detectors. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 143–152, New York, NY, USA, 2000. ACM.

[27] Matthias Fuegger, Ulrich Schmid, Gottfried Fuchs, and Gerald Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proceedings of the Sixth European Dependable Computing Conference (EDCC-6)*, pages 87–96. IEEE Computer Society Press, October 2006.

[28] Josef Widder and Ulrich Schmid. Achieving synchrony without clocks. Research Report 49/2005, Technische Universität Wien, Institut für Technische Informatik, 2005. (to appear in Distributed Computing, 2009).

[29] Martin Biely and Josef Widder. Optimal message-driven implementation of Omega with mute processes. In *Proceedings of the Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2006)*, volume 4280 of *LNCS*, pages 110–121, Dallas, TX, USA, November 2006. Springer Verlag.

[30] Peter Robinson and Ulrich Schmid. The Asynchronous Bounded-Cycle Model. In *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'08)*, volume 5340 of *Lecture Notes in Computer Science*, pages 246–262, Detroit, USA, November 2008. Springer Verlag. (Best Paper Award).

[31] Achour 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.

[32] Achour Mostefaoui, Michel Raynal, and Corentin Travers. Crash-resilient time-free eventual leadership. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems (SRDS 2004)*, pages 208–217. IEEE Computer Society, 2004.

[33] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Chasing the weakest system model for implementing omega and consensus. *IEEE Transactions on Dependable and Secure Computing*. (to appear).

[34] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[35] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Shared Memory vs Message Passing. LPD-REPORT 001, Ecole Polytechnique Federale de Lausanne, 2003.

[36] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.

[37] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[38] Anindya Basu, Bernadette Charron-Bost, and Sam Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In Özalp Babaoğlu, editor, *Distributed algorithms*, volume 1151 of *Lecture Notes in Computer Science*, pages 105–122, Oct 1996.

[39] Ajei Sarat Gopal. *Fault-tolerant broadcasts and multicasts: the problem of inconsistency and contamination*. PhD thesis, Cornell University, Ithaca, NY, USA, 1992.

[40] Rida A. Bazzi and Gil Neiger. Simulating crash failures with many faulty processors (extended abstract). In *WDAG '92: Proceedings of the 6th International Workshop on Distributed Algorithms*, pages 166–184, London, UK, 1992. Springer-Verlag.

[41] Rachid Guerraoui, Maurice Herlihy, Petr Kouznetsov, Nancy Lynch, and Calvin Newport. On the weakest failure detector ever. In *Proceedings of the twenty-sixth annual ACM Symposium on Principles of Distributed Computing (PODC'07*, pages 235–243. ACM, August 2007.

[42] Piotr Berman and Juan A. Garay. Randomized distributed agreement revisited. In *Proceedings 23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23)*, pages 412–419, Toulouse, France, 1993. IEEE Computer Society Press.

[43] Prasad Jayanti and Sam Toueg. Every problem has a weakest failure detector. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing (PODC '08)*, pages 75–84, New York, NY, USA, 2008. ACM.