The Generalized Loneliness Detector and Weak System Models for *k*-Set Agreement

Martin Biely, Peter Robinson, and Ulrich Schmid

Abstract—This paper presents two weak partially synchronous system models $\mathcal{M}^{\mathrm{anti}(n-k)}$ and $\mathcal{M}^{\mathrm{sink}(n-k)}$, which are just strong enough for solving k-set agreement: We introduce the generalized $(n-k)$-loneliness failure detector $\mathcal{L}(k)$, which we first prove to be sufficient for solving k-set agreement, and show that $\mathcal{L}(k)$ but not $\mathcal{L}(k-1)$ can be implemented in both models. $\mathcal{M}^{\mathrm{anti}(n-k)}$ and $\mathcal{M}^{\mathrm{sink}(n-k)}$ are hence the first message passing models that lie between models where Ω (and therefore consensus) can be implemented and the purely asynchronous model. We also address k-set agreement in anonymous systems, that is, in systems where (unique) process identifiers are not available. Since our novel k-set agreement algorithm using $\mathcal{L}(k)$ also works in anonymous systems, it turns out that the loneliness failure detector $\mathcal{L} = \mathcal{L}(n-1)$ introduced by Delporte et al. is also the weakest failure detector for set agreement in anonymous systems. Finally, we analyze the relationship between $\mathcal{L}(k)$ and other failure detectors suitable for solving k-set agreement.

Index Terms—Distributed systems, models of computation

1 Introduction

In recent years, the quest for weak system models (resp. failure detectors [21]), which add just enough synchrony (resp. failure information/fairness [50]) to purely asynchronous systems to circumvent impossibility results for agreement problems, has been an active research topic in distributed computing. Most work in this area falls into one of the following two categories: 1) Finding weak(est) failure detectors and 2) defining weak partially synchronous models that add just enough synchrony to the asynchronous model for solving a given agreement problem—more specifically: consensus, set agreement or k-set agreement. In the *k-set agreement* [23] problem, one considers n processes, each starting with a (possibly different) initial value; (correct) processes must decide on one of the initial values such that no more than k different values are decided upon system-wide. *Set agreement* resp. *consensus* refers to the special case $k = n-1$ resp. $k = 1$ (where all processes have to decide on the same value).

Historically, the first of the aforementioned impossibility results is the FLP result by Fischer et al. [33], which established that consensus among n processes is impossible to solve in asynchronous systems if just $f = 1$ process may crash. Only later it has been shown that similar results hold for the k-set agreement problem in asynchronous systems with up to $f = k$ crashes [17], [39], [53]. In the context of consensus, the eventual leader oracle Ω [20], which eventually outputs the identifier of one correct process everywhere, was identified as the weakest failure detector for solving consensus a) for shared memory systems and b) for message passing systems where a majority of the processes is correct. Research then shifted towards weak partially synchronous models that allow to implement Ω. The first implementation of Ω was provided in [42] and was based on a variant of the partially synchronous model of [30]. The subsequent quest for the weakest synchrony assumptions for implementing Ω was started by [3], and resulted in a series of papers [3], [4], [43], [40], [31] in which the number of required timely links has been reduced considerably. In [40], it was 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, [9] revealed that Ω is sufficient for implementing an eventual $(n-1)$-source. In the most recent paper [31], the intermittent rotating f-star assumption was introduced, which can be seen as a further generalization of the timely f-source assumption.

For message passing systems, Ω was initially only known to be sufficient for consensus when $n > 2f$, whereas for shared memory the result also holds for the wait-free case (i.e., $f = n-1$). The apparent gap was eventually closed by Delporte-Gallet et al. [26], where it was proved that the quorum failure detector Σ is the weakest for implementing shared memory in message passing systems (also in those that allow a majority of the processes to fail). Moreover, the combination of Σ and Ω was shown to be the weakest failure detector for solving consensus for any number of failures in message passing systems. Note that Σ can be implementing in asynchronous message passing systems with a majority of correct processes.

Turning to (k-)set agreement, we note that most of the existing work is devoted to weak failure detectors. In [56], a failure detector called *anti-*Ω was shown to be the weakest

- *M. Biely is with EPFL IC IIF LSR, INF 233 (Bâtiment INF), Station 14, Lausanne 1015, Switzerland. E-mail: martin.biely@epfl.ch.*
- *P. Robinson is with the Division of Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, SPMS MAS #03-01, Singapore 637371. E-mail: peter.robinson@ntu.edu.sg.*
- *U. Schmid is with the Embedded Computing Systems Group, Vienna University of Technology, Treitlstrasse 3, 2nd floor, Wien 1040, Austria. E-mail: s@ecs.tuwien.ac.at.*

Manuscript received 11 Sept. 2012; revised 13 Feb. 2013; accepted 26 Feb. 2013; date of publication 19 Mar. 2013; date of current version 21 Feb. 2014. Recommended for acceptance by R. Baldoni.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.77

for set agreement in shared memory systems [55]. Like Ω, anti-Ω also returns the identifier of some process. The crucial difference to Ω is that anti-Ω eventually never outputs 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, returns $n-k$ processes and has been proved in [32], [35] to be the weakest failure detector for k-set agreement in shared memory systems.

In [28], the "loneliness" failure detector $\mathcal{L}$ was shown to be the weakest failure detector for $(n-1)$-set agreement in message passing systems.

With respect to general k-set agreement, [14], [16] introduced the quorum family Σ_k and proved that it is necessary for solving this problem. The paper also proved that the failure detector family $\Pi_k = \langle \Sigma_k, \Omega_k \rangle$ coincides with the weakest failure detectors $\langle \Sigma, \Omega \rangle$ for $k = 1$, and with $\mathcal{L}$ for $k = n-1$. Herein, Ω_k is a generalization of Ω introduced in [49], which returns sets of k process ids that eventually stabilize and contain a correct process. However, for general values of $2 \leqslant k \leqslant n-2$, it turned out [11], [18], [51] that Π_k is not sufficient and thus not the weakest failure detector for k-set agreement. Thus, the quest for the weakest failure detector for general message passing k-set agreement is still open.

In sharp contrast to the considerable efforts spent on failure detectors for k-set agreement surveyed above, very little is known about partially synchronous models for this problem. Besides some time complexity results in systems where periods of synchrony and asynchrony alternate [5], we are only aware of one related approach (albeit for shared memory systems), namely, the set timeliness approach for k-set agreement in shared memory systems introduced in [1], [2]. Consult Section 3.3 for a more detailed relation of our models & results and existing ones.

This paper introduces both a weak failure detector and weak partially synchronous models for solving k-set agreement in wait-free message passing systems, i.e., where at most $f = n-1$ of the n processes in the system may crash. The detailed contributions are as follows:

a) *The failure detector $\mathcal{L}(k)$*: We introduce the generalized "$(n-k)$-loneliness" failure detector $\mathcal{L}(k)$ in Section 2, which generalizes the loneliness failure detector $\mathcal{L} = \mathcal{L}(n-1)$ from [28]. In Section 4, we show that $\mathcal{L}(k)$ is sufficient for solving k-set agreement, by giving an algorithm and proving it correct. We also establish that there is no algorithm that solves $(k-1)$-set agreement with $\mathcal{L}(k)$.

In Section 5, we compare $\mathcal{L}(k)$ to the limited scope failure detector $\mathcal{S}_{n-k+1}$ with respect to the failure detector hierarchy [21]. For the border 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. We also analyze the relationship of $\mathcal{L}(k)$ to the quorum failure detector Σ and its generalization Σ_k, which is known to be necessary for solving k-set agreement [16] and hence weaker than $\mathcal{L}(k)$.

b) *Weak system models*: In Section 3, we introduce two novel system models $\mathcal{M}^{\mathrm{anti}(n-k)}$ and $\mathcal{M}^{\mathrm{sink}(n-k)}$, which allow to implement $\mathcal{L}(k)$ and hence to solve k-set agreement. Model $\mathcal{M}^{\mathrm{anti}(x)}$ is a time-free model based on expressing synchrony via message ordering properties, whereas model $\mathcal{M}^{\mathrm{sink}(x)}$ is similar in spirit to partially synchronous models like [3], [30], [50]. We also prove that $\mathcal{M}^{\mathrm{anti}(n-k)}$ and $\mathcal{M}^{\mathrm{sink}(n-k)}$ are too weak for solving $(k-1)$-set agreement. To the best of our knowledge, these models are hence the first message passing models that provide just enough synchrony to solve k-set agreement, but no stronger agreement problem (including consensus). Note that we also show that neither $\mathcal{M}^{\mathrm{anti}(n-k)}$ nor $\mathcal{M}^{\mathrm{sink}(n-k)}$ is strong enough to implement the limited scope failure detector $\mathcal{S}_{n-k+1}$, which is known to be sufficient for implementing k-set agreement [46]. This indicates that the models are closer to what is necessary for k-set agreement than models that allow implementing $\mathcal{S}_{n-k+1}$.

c) *Anonymous systems*: In Section 6, we turn our attention to anonymous systems (without (unique) process identifiers). We explain how to derive anonymous versions of our system models and introduce an $\mathcal{L}(k)$-based k-set agreement algorithm. As it does not use process ids, it follows from [28] that $\mathcal{L}$ is also the weakest failure detector for set agreement in anonymous systems. Finally, we discuss the relation between $\mathcal{L}(1)$ and failure detectors AP and $A\Omega$, and its impact on the quest for the weakest failure detector for consensus in anonymous systems.

For conciseness, a number of definitions, results and proofs have been relegated to the online supplement [12] of this paper.

2 System Models and Problem Definition

The models we consider in this paper are based on the standard asynchronous model of [33], which we denote by $\mathcal{M}^{\mathrm{async}}$: It comprises a set Π of n distributed processes, which communicate via message passing over a fully-connected point-to-point network made-up of pairs of unidirectional links with finite but unbounded message delays. Links need not be FIFO but are assumed to be reliable.[1] Every process executes an instance of a distributed algorithm and is modeled as a deterministic state machine. Its execution consists of a sequence of instantaneous local *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 with time instants taken from the infinite set T. Whenever, a process takes a step the clock ticks (i.e., it advances by one time unit). Note that processes do not have access to this clock. For simplicity, we assume that $T = \mathbb{N}$.

A correct process is correct if it takes infinitely many steps in a run. The algorithm run by a process can *halt* by entering a terminal state, in which it remains for infinitely many steps (receiving but discarding all messages sent to it). By contrast, a faulty process is one that takes only a finite number of steps. In its last step, a process can omit

1. In [12, Section II.A], we briefly discuss relaxations of this assumption.

to send some, but not all, messages it is required to send by its code.[2] We call a process *alive* at time t if it takes a step at or after t, and *crashed* otherwise.

The *failure pattern* of α is a function $F : T \to 2^{\Pi}$ that outputs the set of crashed processes at 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, i.e., we consider the wait-free [38] case.

A run α is *admissible in* $\mathcal{M}^{\text{async}}$ if 1) a message is only received at time t by process p if it was sent by some process q to it at some time $t' \leqslant t$, and 2) every message sent to p is eventually received if p is correct.

2.1 k-Set Agreement

In the k-set agreement problem [23], every process starts with a proposal value v from a finite[3] domain V and must eventually irrevocably decide on some value as follows:

k-Agreement:	Processes must decide on atmost k different values system-wide.
Validity:	If a correct process decides on v, then v was proposed by some process.
Termination:	Every correct process must eventually decide.

For $k = n-1$, the problem is also referred to as set agreement, whereas $k = 1$ is equivalent to uniform consensus [22] (as the agreement property ties together the decision values of both correct and faulty processes). Note that it is well known that k-set agreement is impossible in purely asynchronous systems when $f \geqslant k$ processes might crash [17], [39], [53].

2.2 Failure Detectors

A failure detector [21] $\mathcal{D}$ 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 $\mathcal{D}$.

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 some algorithm $A_{\mathcal{D}\to\mathcal{D}'}$ transforms $\mathcal{D}$ to $\mathcal{D}'$, if for any run of an asynchronous system equipped with D (with failure pattern F) it maintains an output variable $output_{D'}$ that simulates legal (admissible for F) failure detector histories of $\mathcal{D}'$ at every process. We say that $\mathcal{D}'$ is *weaker* than $\mathcal{D}$ and say $\mathcal{D}$ is *stronger* than $\mathcal{D}'$, if such an algorithm $A_{\mathcal{D}\to\mathcal{D}'}$ exists. 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 *weakest for problem* P if $\mathcal{D}$ is weaker than any failure detector $\mathcal{D}$ that solves P.

Recently, it was shown in [28] that the "loneliness"-detector $\mathcal{L}$ is the weakest failure detector for message passing set agreement. Intuitively speaking, there is (at least) one possibly faulty 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 introduced in [10], which we denote by $\mathcal{L}(k)$ (with $\mathcal{L} = \mathcal{L}(n-1)$). Instead of loneliness, it enables processes to detect "$(n-k)$-loneliness". Formally speaking, a process p is *$(n-k)$-lonely at time t in a run α*, if $p \notin F(t)$ and $|F(t)| \geqslant k$ in α.

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:*

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

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

Before discussing other failure detectors for k-set agreement, it is worthwhile to recall that a failure detector is called *realistic* [24] if and only if it can be implemented in a synchronous system with $f = n-1$; otherwise it is *non-realistic*. Moreover, we say that a model $\mathcal{M}$ is *non-realistic* if a non-realistic failure detector can be implemented in $\mathcal{M}$. Mostéfaoui et al. [47, Theorem 2]] have shown that $k \geqslant n/2$ is a necessary and sufficient condition for $\mathcal{L}(k)$ to be realistic.

Another class of failure detectors for k-set agreement are the limited scope failure detectors introduced in [37], [45], which output sets of process ids. Such failure detectors have the strong completeness property of the *strong* failure detector $\mathcal{S}$ [21], but their accuracy is limited to a set of processes called the scope; see [12, Definition 1]. In the special case where the scope comprises all processes, $\mathcal{S}_n$ coincides with $\mathcal{S}$.[4] It was shown in [46] that $\mathcal{S}_{n-k+1}$ is sufficient for k-set agreement.

While the weakest failure detector for message passing k-set agreement is still unknown, Bonnet and Raynal have introduced the *quorum family* Σ_k in [16] (see [12, Definition 2]), and shown that Σ_k is necessary for solving k-set agreement, i.e., that any failure detector $\mathcal{X}$ that allows to solve k-set agreement can be transformed into Σ_k.

3 Weak System Models for Set Agreement

In this section, we introduce two system models $\mathcal{M}^{\text{anti}(x)}$ and $\mathcal{M}^{\text{sink}(x)}$, which restrict the set of admissible runs in $\mathcal{M}^{\text{async}}$ by weak synchrony conditions. By implementing $\mathcal{L}(k)$ in both $\mathcal{M}^{\text{anti}(n-k)}$ and $\mathcal{M}^{\text{sink}(n-k)}$, we show that they are strong enough for solving k-set agreement. The models differ in the way how synchrony properties are added: $\mathcal{M}^{\text{anti}(x)}$ uses

2. This definition allows "unclean" crashes. The assumption that at least one message is sent in the last step is for simplicity only: A crash that causes all messages to be lost is modeled as a crash before this step.

3. We assume that $|V| \geqslant n$.

4. For the case $k > f$ (which is not relevant here as $f = n-1$) [45] also provides a transformation $T_{\mathcal{S}_k \to \mathcal{S}}$.

message-ordering and is hence time-free, whereas $\mathcal{M}^{\mathrm{sink}(x)}$ relies on classical partial synchrony assumptions [30].

3.1 The Model $M^{\mathbf{anti}(x)}$

In some application domains, e.g., systems-on-chip [34], message-driven execution models [13], [52], [54], where computing steps are triggered by the arrival of messages instead of the passage of time, can be advantageous over the usual time-driven execution model. The main advantage here is that there is no need for dedicated clocks to trigger steps. For a more detailed discussion of this issue, we refer the interested reader to [54, Section 5].

The model $\mathcal{M}^{\mathrm{anti}(x)}$ presented in this section belongs to the aforementioned category of message-driven models. Inspired by the round-trip-based model introduced in [44], [48], we specify our synchrony requirements as conditions on the order of round-trip message arrivals. Computations proceed in asynchronous local (i.e., uncoordinated) rounds: At the start of its round, process p sends a $(query)$-message[5] to all processes, including itself. If a process receives a $(query)$-message from some process q, it sends a $(response)$-message to q. Once a round ends, all further responses to the query are discarded by the system. Clearly, such behaviour could be implemented by attaching sequence numbers to all queries.

In [44], a round ends when $n - f$ responses have been received. In a wait-free setting like ours, this means that a round ends when the first response arrives. By contrast, in $\mathcal{M}^{\mathrm{anti}(x)}$, a round ends when a process receives its own response: This triggers an $end{-}round$ event, upon which the process obtains the set of all processes that have responded in this round.

The synchrony condition of our model is encapsulated in the central concept of an x-anti-source:

Definition 2 (x-Anti-Source). *A correct or faulty process p is an x-*anti-source*, if, whenever p sends a* (query) *to all remote processes it will receive* (response)*-messages from at least x remote processes before process p starts a new round.*

As discussed in considerable detail in [54], such a time-free specification does not mean that there is also a time-free *implementation* of an x-anti-source. Still, bounded link delay *ratios*, rather than bounded absolute delay values, are sufficient for asserting the existence of an x-anti-source.

Note that an anti-source, as we have previously introduced it [10], is not the same as a 1-anti-source: In the former, a round ended after receiving the first response like in [44]. Thus, an anti-source was defined as a process whose round-trips with itself is never the fastest. The subtle difference is that a 1-anti-source can receive any number (up to n) of $(response)$-messages, while an anti-source can only receive one. Moreover, a 1-anti-source will always see its own $(response)$ while this is not the case for an anti-source. Fig. 1 shows an example execution where process p is a 1-anti-source. That is, in Fig. 1, an algorithm running on p will—besides its own response—only see the response of p_1 for the first request and only the response of p_2 for the second request.

5. In [44], query and response messages are allowed to carry additional data. We omit this possibility here, since it is not needed to implement $\mathcal{L}(k)$.

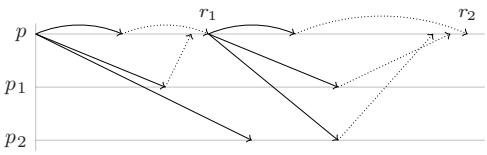

Fig. 1. An execution of an algorithm in model $\mathcal{M}^{\mathrm{anti}(x)}$ where process p is a 1-anti-source. For the sake of readability, we have included only the messages p sees by the end of its rounds. That is, we omitted the late responses as well as $(query)$-messages sent by p_1 and p_2 and the corresponding $(response)$-messages.

Definition 3. *Let α be a run of a distributed algorithm. Then, α is* admissible in $\mathcal{M}^{\mathrm{anti}(x)}$ *if the following holds:*

1. *Run α is admissible in $\mathcal{M}^{\mathrm{async}}$.*
2. *At least x processes are an x-anti-source in α.*

3.1.1 Detecting k-Loneliness in Model $\mathcal{M}^{\mathrm{anti}(k)}$

Algorithm 1 provides an implementation of the k-loneliness failure detector $\mathcal{L}(n-k)$ in $\mathcal{M}^{\mathrm{anti}(k)}$. A process sets its $output_{\mathcal{L}}$ to TRUE if and only if it receives responses from less than k remote processes. A process that is a k-anti-source will thus never change its variable $output_{\mathcal{L}}$ to TRUE.

Algorithm 1 $\mathcal{L}(n-k)$ in Model $\mathcal{M}^{\mathrm{anti}(k)}$

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

6:  upon end of round r do
7:      /* R is the set of remote processes we got responses from */
8:      if |R| ⩾ k then
9:          startRound()
10:     else
11:         output_L ← TRUE

12: upon receive (query) from q do
13:     send (response) do q

14: procedure startRound()
15:     send (query) to all
```

Theorem 1. *$\mathcal{L}(n-k)$ is implementable in $\mathcal{M}^{\mathrm{anti}(k)}$.*

Proof. Let p be a k-anti-source in a run of Algorithm 1. At the start of every round, process p sends a $(query)$-message tagged with the round number to all other processes. By the definition of a k-anti-source, p never receives $(response)$-messages from less than k remote processes. Process p will therefore always pass the test in Line 8 and start a new round. It follows that p never reaches Line 11, that is, its $output_{\mathcal{L}}$ remains FALSE forever, which entails Property (1) in Definition 1.

To show Property (2), consider a run α where at least $n-k$ processes crash. Let q be a correct process in α. Then, there is a time after which all faulty processes have crashed, and thus cannot respond to q's query. That is, there will be some query such that q will get only responses from other correct processes. As there are less than k such processes, clearly, the test in Line 8 will fail and q will set $output_{\mathcal{L}}$ to TRUE once and forever in Line 11. □

3.1.2 Discussion of $\mathcal{M}^{\text{anti}(x)}$

As shown in [47, Theorem 2], $\mathcal{L}(k)$ is not realistic, i.e., it cannot be implemented in a synchronous system with up to $n-1$ crash failures if $2k < n$. With respect to $\mathcal{L}(n-k)$, this translates into the condition $2(n-k) < n$ and hence $n < 2k$ for $\mathcal{L}(n-k)$ not being realistic. Since $\mathcal{L}(n-k)$ *can* be implemented in $\mathcal{M}^{\text{anti}(k)}$, this sheds some light on the relation between $\mathcal{M}^{\text{anti}(k)}$ and the wait-free synchronous model.

We start our considerations with the following simple observation: If a process p is a 1-anti-source, it cannot be the only correct process, because after all other processes have crashed, no process remains to guarantee that it will receive a response from a remote process to each query. In general, no k-anti-source can be among the last k alive processes. Now, consider $\mathcal{M}^{\text{anti}(k)}$ for $n < 2k$: The model requires the existence of k processes that are k-anti-sources. As long as some k-anti-source is alive, it requires k other processes to be alive as well. Consequently, i) if only $i \leqslant k$ processes are alive at some point in a run, then none of them can be a k-anti source. On the other hand, ii) when only $j < k$ processes have crashed at some point in a run, then least one of the k required k-anti-sources must be alive. When $n = i + j < 2k$, then these two contradicting cases can happen simultaneously at some time: This happens in every run where the number of alive processes ever becomes $\leqslant k$, although the number of crashed processes is still $< k$. This gives raise to the following observation:

Observation 1. When $n < 2k$, there are no admissible runs in $\mathcal{M}^{\text{anti}(k)}$ where $i \leqslant k$ processes are correct and $n - i < k$ are faulty.

On the other hand, in a run where at least $f' \geqslant k$ processes crash, less than k additional processes can remain alive in case of $n < 2k$. Hence, the only runs where i) and ii) are guaranteed not to hold true simultaneously at some time in case of $f' \geqslant k$ are "trivial" admissible runs with at least k initially dead k-anti-sources.

Therefore, it turns out that $\mathcal{M}^{\text{anti}(k)}$ and the synchronous model with $f = n-1$ are incomparable in case of $n < 2k$: On the one hand, the synchronous model is obviously stronger than $\mathcal{M}^{\text{anti}(k)}$ since it requires *every* process to receive all messages from correct processes in a round. On the other hand, $\mathcal{M}^{\text{anti}(k)}$ is stronger than the wait-free synchronous model, since the failure bound $f = n-1$ cannot be "tight" in a non-trivial run of $\mathcal{M}^{\text{anti}(k)}$: The existence of just one not initially dead k-anti-source does not allow more than $f = k-2$ crash failures.

Observation 2. Model $\mathcal{M}^{\text{anti}(k)}$ for $n < 2k$ (and hence $\mathcal{M}^{\text{anti}(n-k)}$ for $2k < n$) are non-realistic, in the sense that there are runs in the synchronous wait-free model that are not admissible in $\mathcal{M}^{\text{anti}(k)}$.

All runs in a synchronous system with $n < 2k$ processes where at most $f = k-2$ can crash are admissible in $\mathcal{M}^{\text{anti}(k)}$, however.

3.2 The Model $\mathcal{M}^{\text{sink}(x)}$

The model $\mathcal{M}^{\text{sink}(x)}$ is a weak variant of the classic partially synchronous models [29], [30], as are the weak-timely link (WTL) models [3], [4], [40], [43]. Essentially, all those models assume that processes are partially synchronous and try to minimize the synchrony requirements on communication delays.

In the model $\mathcal{M}^{\text{anti}(x)}$ introduced before, there is no time bound on the duration of a round-trip, as only the arrival order or response messages matters. Our second model $\mathcal{M}^{\text{sink}(x)}$ enforces a similar ordering by means of explicit communication delay bounds and message timeouts. 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 timely link from one process. This assumption would make one process permanently 1-*accessible* (in the notation of [43]), though, which is unnecessarily strong.

As in [29], 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, i.e., the number of steps processes can take during the transmission of m. We say that a message m is delivered *timely* over the link (p, q) iff it is received by q not after q has taken Δ steps after p has sent the message.[6] Note that this definition implies that all messages sent to a crashed process (or a process that crashes before taking Δ steps) are considered to be delivered timely.

Although we use Δ and Φ to describe synchrony as in [29], [30], $\mathcal{M}^{\text{sink}(x)}$ differs from these models w.r.t. the atomicity of steps: We assume that processes can both receive and broadcast (i.e., send multiple messages) in the same step. Although this assumption is not really vital, it considerably simplifies our algorithm and its proof: As Algorithm 2 employs step-counting and asynchronous rounds, we would otherwise have to argue about a process being in the middle of broadcasting a message (which would take n steps, i.e., up to $n\Phi$ time), which complicates the definition and analysis of round switching.

Algorithm 2 $\mathcal{L}(n-k)$ in Model $\mathcal{M}^{\text{sink}(k)}$

```
1:  Vars:
2:  output_L ∈ {TRUE, FALSE}
3:  seen[ℕ] ∈ ℤ
4:  Initially:
5:  output_L ← FALSE
6:  ∀i ∈ ℕ : seen[i] ← 0
7:  startPhase(0)

8:  in step number (i · η) do
9:      startPhase(i)
10: in step number ((i + 1) · Φη − 1 + Δ) do
11:     if seen[i] < k then
12:         output_L ← TRUE
13: upon receive (alive, ph) do
14:     seen[ph] ← seen[ph] + 1

15: procedure startPhase(phase)
16:     send (alive, phase) to all remote
```

As in the WTL models (and in contrast to [29], [30]), we do not assume Δ to hold for all messages. Rather, we base our synchrony conditions on "sinks", i.e., processes that can always receive some messages timely.

6. Note that Δ refers to steps of the receiver here, as in [29], [50] and contrasting [30], where Δ is measured in some notion of real-time.

Definition 4 (x-Sink). *A correct or faulty process q is an x-*sink *in a run α, if there is a set P of $|P| = x$ processes, which do not crash earlier than q, such that any message sent by $p \in P$ to q is delivered timely to q.*

Observe that the decisive difference between q being a x-sink and p being a perpetual 1-source (in the notation of [3]) is that q may crash in our model.

Note that we implicitly assume that all processes initially start-up at the same time ($T = 0$) as usual. All our reasoning below would also apply, however, if we allowed sinks to start-up later: All that is actually needed is that no message from p is successfully received by q after Δ time.

Definition 4 is not the end of the road, however, as this synchrony requirement can be further weakened in case of 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 with heartbeat-based failure detectors. For such algorithms, we also provide an alternative (and in fact even weaker) definition of an x-sink, where the timely processes P may change. It is similar in spirit to the timely f-source model with moving timely links [40], albeit complicated by the fact that we cannot rely on a single (send-) event as a common reference point here. For $i \geqslant 1$, let $A(i)$ be the set of (alive) processes that perform the ith broadcast step (at least partially, in case of a crash during the step) starting round i. Moreover, let $\sigma(i)$ be the time (according to our global clock) of the step in which the last process performs its ith broadcast.

Definition 5 (x-Sink'). *A process q is an x-*sink' *in a run α if for every $i \geqslant 1$ it holds that*

a. if $|A(i) \setminus \{q\}| \geqslant x$, then there is a set $P(i)$ with $|P(i)| \geqslant x$ such that $\forall p \in P(i)$, q timely receives the ith message sent by p.
b. Otherwise, if $|A(i) \setminus \{q\}| < x$, then q does not take Δ or more steps after $\sigma(i)$.

As a message is—per definition—timely when sent to a process that does less than Δ steps after the message was sent, we have that initially dead processes are always x-sinks'. Moreover, item (b) ensures that an x-sink is never able to "time out" more than $n - x$ remote processes, even if not enough messages are sent that could arrive timely.

Definition 6 (Model $\mathcal{M}^{\mathrm{sink}(x)}$). *Let α be a run of a distributed algorithm. Then, α is* admissible in $\mathcal{M}^{\mathrm{sink}(x)}$ *if the following holds:*

1. *Run α is admissible in $\mathcal{M}^{\mathrm{async}}$.*
2. *There is a bound Φ, such that every correct process takes at least one step in any interval of time containing Φ steps of any other process. Moreover, there is a bound Δ on the maximum number of steps taken by any process during the transmission of a timely message.*
3. *At least x processes are x-sinks in α.*

Note that, in contrast to $\mathcal{M}^{\mathrm{anti}(x)}$, the model $\mathcal{M}^{\mathrm{sink}(1)}$ is equivalent to the $\mathcal{M}^{\mathrm{sink}}$ model introduced in [10].

At a first glance, it might be surprising that model $\mathcal{M}^{\mathrm{sink}(x)}$ 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 $\mathcal{L}(k)$ (see Definition 1), however, 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 also involves the non-eventual Σ (see [25]).

The non-eventuality of $\mathcal{L}(k)$ also implies that the model parameters Φ and Δ must be known and have to hold right from the start: After all, $n - k$ processes, namely, all the $(n-k)$-sinks in $\mathcal{M}^{\mathrm{sink}(n-k)}$, must never falsely suspect $(n-k)$-loneliness and set their output to TRUE, as this would otherwise violate (1). Although it would be sufficient if only the x-sinks knew the model parameters Φ and Δ, we do not assume that the sinks are known in advance, so all processes must know these parameters.

3.2.1 Detecting k-Loneliness in Model $\mathcal{M}^{\mathrm{sink}(k)}$

Algorithm 2 shows a simple protocol that implements $\mathcal{L}(n-k)$ in model $\mathcal{M}^{\mathrm{sink}(k)}$. Variable $output_{\mathcal{L}}$ contains the simulated failure detector output. Every process p periodically (every η steps) sends out $(alive, phase)$-messages that carry the current phase-counter i. In addition, p keeps track of the number of $(alive, ph)$-messages received from different other processes in the array $seen[ph]$. In case it did not receive at least k messages by the end of the current phase i in Line 10, it sets $output_{\mathcal{L}} \leftarrow$ TRUE in Line 12.

Theorem 2. *Algorithm 2 implements failure detector $\mathcal{L}(n-k)$ in model $\mathcal{M}^{\mathrm{sink}(k)}$ for $f = n - 1$.*

3.3 Discussion of $\mathcal{M}^{\mathrm{sink}(x)}$

As detailed in [12], the k-sinks used in $\mathcal{M}^{\mathrm{sink}(k)}$ provide considerably weaker communication synchrony than (perpetual) k-sources, as used in the WTL model [4], [40]. Like $\mathcal{M}^{\mathrm{anti}(k)}$, $\mathcal{M}^{\mathrm{sink}(k)}$ is also non-realistic for $n < 2k$ (and hence $\mathcal{M}^{\mathrm{sink}(n-k)}$ for $2k < n$), though.

Interestingly, the fact that $\mathcal{L}(k)$ and hence k-set agreement can be implemented in $\mathcal{M}^{\mathrm{sink}(n-k)}$ is also in accordance with results obtained in the generalized set timeliness model of [2], despite the fact that the latter has been devised for shared memory systems. As opposed to classic partially synchronous processes [29], where every individual process must be timely with respect to every other individual process, in the sense that it makes at least one step when the other process made Φ steps, [2] requires such a property only for *sets* of processes: In model $\mathcal{S}^i_{j,n}$, there must be at least one set I of size i that is timely with respect to some other set J of size j, in the sense that within Φ steps of (possibly different) processes in J, some process in I must make at least one step. The authors proved that k-set agreement is solvable in $\mathcal{S}^i_{j,n}$, in the presence of up to f crash failures, iff $i \leqslant k$ and $j - i \geqslant f + 1 - k$.

Now consider $\mathcal{M}^{\mathrm{sink}(n-k)}$, which guarantees $n - k$ processes $q_1, \ldots, q_{n-k}$ that act as $(n-k)$-sinks, i.e., receive timely the messages from at least $n - k$ other processes. In order to do so, every q_i must receive timely from at least one process outside $\{q_1, \ldots, q_{n-k}\}$. This is in accordance with the findings of [2], if one considers $J = \Pi$ (which includes the $n - k$-sinks) and $I = \Pi \setminus \{q_1, \ldots, q_{n-k}\}$: Since $i = k$ and $j = n$ here, the above equations tell that k-set agreement is solvable; since one can implement $\mathcal{L}(k)$ in $\mathcal{M}^{\mathrm{sink}(n-k)}$, this is indeed in accordance with our findings.

3.4 k-Set Agreement Impossibility

In Section 4, we provide an algorithm that solves k-set agreement with $\mathcal{L}(k)$, which in turn is implementable in $\mathcal{M}^{\text{anti}(n-k)}$ (Theorem 1) and $\mathcal{M}^{\text{sink}(n-k)}$ (Theorem 2). In this section, we will show that it is impossible to solve k'-set agreement for $k' < k$ in either model. Note carefully that this result reveals that the parameter k precisely characterizes the k-set agreement solvability border in both $\mathcal{M}^{\text{anti}(n-k)}$ and $\mathcal{M}^{\text{sink}(n-k)}$, for every choice of n (both models are non-realistic for $2k < n$, however).

Theorem 3. *It is impossible to solve k-set agreement among n processes in $\mathcal{M}^{\text{anti}(x)}$ for $k \leqslant n - x - 1$.*

Proof. Assume, for the sake of a contradiction, that some algorithm A solves this problem and consider runs of A where the x required x-anti-sources are initially dead. Since there are no further synchrony requirements in $\mathcal{M}^{\text{anti}(x)}$, all remaining processes can communicate totally asynchronously. Thus, there is a one-to-one relationship between these runs and the runs A produces in an asynchronous system of $n' = n - x$ processes of which $f' = n' - 1 = n - x - 1 =$ processes can crash. However, due to the k-set impossibility results of [17], [39], [53], there is no algorithm that solves k-set agreement in an asynchronous system where f' out of n' processes may crash in case of $k \leqslant f'$. □

The analogous result for $\mathcal{M}^{\text{sink}(x)}$ is provided in [12].

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

In this section, we present an algorithm that solves k-set agreement in an (anonymous) asynchronous system augmented with a failure detector $\mathcal{L}(k)$. In addition, we prove that it is impossible to solve $(k-1)$-set agreement with $\mathcal{L}(k)$.

The algorithm for solving $(n-1)$-set agreement with $\mathcal{L}$ presented in [28] requires a total order on process identifiers. By contrast, our Algorithm 3 solves k-set agreement for any $1 \leqslant k < n$ on top of $\mathcal{L}(k)$ and works even in anonymous systems. Note carefully that this means that processes neither need to know the unique id of the sender of a message nor that they need to be able to distinguish messages from different senders.

Algorithm 3 proceeds in asynchronous rounds, made up of one or more computing steps that comprise receiving zero or more messages, querying the failure detector ($\mathcal{L}(k)$ in our case), doing some local computation, and optionally broadcasting a message. The messages sent by a process in our algorithm contain the current estimate x and the current round number. In every round r, $0 \leqslant r \leqslant k+1$, every process p that has not yet decided queries its failure detector and decides if $\mathcal{L}(k)$ outputs TRUE. Otherwise, p checks if it has received $n-k$ round r messages from remote processes; note that all the checks are non-blocking and are checked anew in every step. If so, p updates its current estimate x to the minimum of the received values.

At a first glance, it appears counterintuitive that processes terminate after $k+2$ rounds. After all, it would be reasonable to expect that harder agreement tasks like consensus require *more* rounds than, for example, $(n-1)$-set agreement. The reason why this is not the case here is that $\mathcal{L}(k)$ itself becomes much weaker for values of k close to $n-1$, since there are less processes that perpetually output false. To argue informally why $k+2$ rounds are required by our algorithm, consider an execution where in every round $r < k$ (i.e., within k rounds), exactly one process decides in Line 9 and hence no longer participates in later rounds. Thus, $r+1$ processes decide by the end of round $r < k$ via Line 9, and r processes do no longer participate in round $r \leqslant k$. As we will prove in Lemma 4 below, in round r, the remaining processes could decide on at most $k-r$ different values (either in Line 9 or 20) in this case. If the algorithm had a loop bound less than $k+1$, i.e., terminated at the end of some round $r \leqslant k$, we could end up with $r+1+k-r = k+1$ decision values. On the other hand, deciding at the end of round $k+1$ is safe, since no more than k processes can decide via Line 9 as $\mathcal{L}(k)$ can output TRUE at no more than k processes.

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

```
1:  in the first step:
2:  x ← v
3:  rnd ← 0
4:  send (ROUND, 0, x) to all remote processes

5:  in any later step:
6:  receive messages
7:  if L(k) = TRUE then
8:      send (DEC, x) to all remote processes
9:      decide x
10:     halt
11: else if received (DEC, y) then
12:     send (DEC, y) to all remote processes
13:     decide y
14:     halt
15: else if received n − k (ROUND, rnd, y_i)-msgs then
16:     S ← {y_1, ..., y_{n−k}} ∪ {x}
17:     x ← min(S)
18:     if rnd = k + 1 then
19:         send (DEC, x) to all remote processes
20:         decide x
21:         halt
22:     rnd ← rnd + 1
23:     send (ROUND, rnd, x) to all remote processes
```

4.1 Proof of Correctness

We denote by X^r the possibly empty array containing all x-values of processes in the system (with repetitions allowed) that completed the assignment in Line 17 while $rnd = r$. Note carefully that the x-value of a process q that decides via Line 9 in round r is not contained in X^r; this does not imply, however, that no process p could decide on the same decision value as q (after all, repetitions are allowed). 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 unique values in X^r by u^r. If no process reaches Line 17 in round r, the array is empty and both $|X^r|$ and u^r are zero.

Lemma 4. *For any round $r \geqslant 0$, the number of unique values in X^r satisfies $u^r \leqslant k - a_r$, 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, that is, p knows about $n-k+1$ values.

Let p be the process which assigns the largest value x' in Line 17. Since process p computes x' as the minimum of the $n-k+1$ round r values in the multi-set S, it must consist of $n-k+1$ values $y \geqslant x$.

Considering that $|X^r| \leqslant n-a_r$, it follows from $S \subseteq X^r$ that only $n-a_r-(n-k+1) \leqslant k-a_r-1$ values in X^r can be strictly smaller than x'. Thus, processes assign at most $k-a_r$ different values to x in Line 17 (and subsequently send them as $(\text{ROUND}, r+1, x)$-messages). □

Lemma 5. *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 13) 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 9) and due to having received $n-k$ messages in round $k+1$ (Line 20). 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 $rnd = r$. Thus, the number of processes that have decided in Line 9 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 a_r of processes which do not send round r messages is at least σ^{r-1}. Thus, Lemma 4 tells us that $u^r \leqslant k-\sigma^{r-1}$. Now assume by contradiction that there are actually $D > k$ different decisions, with $D \leqslant u^{k+1}+\sigma^{k+1}$. Note that that D is the number of different values decided on in Line 20 plus those that ever decided based on $\mathcal{L}(k)$, which is obviously at most $u^{k+1}+\sigma^{k+1}$. It can be less, since processes need of course not decide on different values. Thus we get $u^{k+1} > k-\sigma^{k+1}$, and by using the above property of u^r for $r = k+1$, 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 implies $D \leqslant u^k+\sigma^k$, as obviously $x \in X^{k+1} \Rightarrow x \in X^k$. We can repeat this argument until we reach $D \leqslant u^1+\sigma^0 = u^1+\ell_0$. Here, Lemma 4 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 $\ell_r \geqslant 1$ 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 $\mathcal{L}(k)$, however, thus proving Lemma 5. □

Theorem 6. *Algorithm 3 solves k-set agreement in an asynchronous system without unique process ids augmented with $\mathcal{L}(k)$.*

Proof. It is immediately apparent from the code that the algorithm also works in anonymous systems, since every process sends its round-r message at most once. *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 5, and since either $n-k$ processes send messages in each round or some process has $\mathcal{L}(k) = \text{TRUE}$, every correct process *terminates*. □

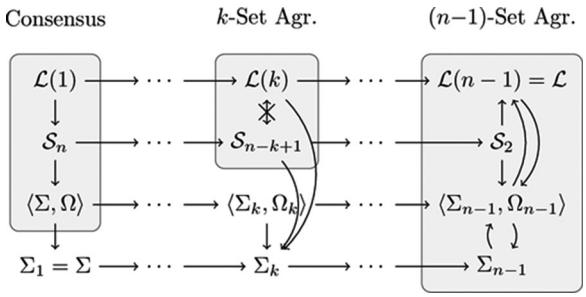

Fig. 2. Failure detector classes for wait-free k-set agreement. A unidirectional arrow from X to Y indicates that failure detector X is stronger than Y; note that this relation is transitive. Arrows in both directions correspond to equality, while the crossed-out arrows indicate incomparability. Failure detectors located within shaded boxes are sufficiently strong for solving the k-set instance given in the column header. The middle column thus shows the "solvability gap," where the (currently) unknown weakest failure detector will fit in.

Theorem 6 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 7. *No algorithm can solve $(k-1)$-set agreement with $\mathcal{L}(k)$, for any $2 \leqslant k \leqslant n-1$.*

Proof. We assume for the sake of a contradiction that such an algorithm A exists. Now consider the failure detector history where $\mathcal{L}(k)$ outputs TRUE at processes $\Pi^t = \{p_1, \ldots, p_k\}$, while it outputs FALSE at the $n-k$ processes $\Pi^f = \{p_{k+1}, \ldots, p_n\}$. This defines a legal history for $\mathcal{L}(k)$ in any run where one of the processes in Π^t is correct. For our proof we now consider the set of runs where all processes in Π^f crash initially. Let this set be R. Now we consider the set of runs of A in an asynchronous system consisting of the k processes $p_1, \ldots, p_k$ equipped with the *dummy* failure detector [36] that always outputs TRUE. Let this set be S. Due to the impossibility of solving set agreement in the asynchronous system [17], [39], [53], A cannot solve $(k-1)$-set agreement in all runs in S. Take any such run ε. Clearly, ε is indistinguishable to some run in R to all processes in Π^t. Thus, A cannot solve $(k-1)$ set agreement in all runs in R, that is, in the asynchronous system augmented with $\mathcal{L}(k)$. □

5 Relation between $\mathcal{L}(k)$ and Other Failure Detectors

In [12, Section 4], we analyze how the $\mathcal{L}(k)$ failure detector relates to some important other failure detectors for message passing k-set agreement. The results are presented in Fig. 2.

6 $\mathcal{L}(k)$ in Anonymous Systems

In this section, we will focus on *anonymous* systems, where processes do not have unique identifiers but can at most distinguish their neighbors via local port numbers, cp. [6], [8], or can distinguish multiple copies of the

same message by other means, i.e., are *numerate* in the notion of [27]. Failure detectors for anonymous resp. homonymous system (where processes may share the same id) have been studied in [15] resp. [7]; a weakest failure detector for consensus has been given in [19].

6.1 Implementing $\mathcal{L}(k)$

Given that we have provided an algorithm that solves k-set agreement using $\mathcal{L}(k)$ without the need for unique identifiers, one natural question to ask is whether this also applies for our algorithms implementing $\mathcal{L}(k)$.

For Algorithm 2, we note that it just counts the number of $(alive, ph)$ messages for each phase ph. So as long as line 13 is triggered by each (identical) message, the algorithm also works in anonymous systems. Moreover, since it does *not* require the knowledge of n, it is also a *uniform* algorithm [15], as long as k does not depend on n. Given that Algorithm 2 implements $\mathcal{L}(n-k)$, assuming this independence might seem self-contradictory. However, this contradiction disappears when considering the aggregate of Algorithm 2 and Algorithm 3 together.

Some considerations related to Algorithm 1 for $\mathcal{M}^{\mathrm{anti}(x)}$ can be found in [12, Section 4].

6.2 Relations to Anonymous Failure Detectors

From [28], we know that $\mathcal{L}$ can be extracted anonymously from any failure detector D that solves set agreement using some algorithm A: Every process executes an independent instance of A (without any other process participating) using D as its failure detector. The simulated $\mathcal{L}$ outputs TRUE at p only when A has terminated at p. In conjunction with our Theorem 6 applied for $k = n-1$, this implies the following fact:

Corollary 8. *$\mathcal{L}$ is the weakest failure detector for set agreement in anonymous message passing systems.*

With respect to consensus, [15] provided an in-depth analysis of various failure detectors for anonymous systems, in particular, the *identity-free perfect failure detector* AP and the *identity-free eventual leader oracle* $A\Omega$, see [12, Definitions 3 and 4].

In [15], it was conjectured that $(A\Sigma, A\Omega) \oplus AP$ is the weakest failure detector for solving anonymous consensus. This *$\oplus$-combination* is defined as the failure detector that outputs $\perp$ for an arbitrary finite prefix and then chooses an output that is admissible for *either* $(A\Sigma, A\Omega)$ or AP at every process.

In [12, Section 4], we disprove this conjecture by showing that $(A\Sigma, A\Omega) \oplus AP$ cannot be extracted from $\mathcal{L}(1)$.

Theorem 9. *Consider an anonymous asynchronous system of at least three processes. Failure detectors $(A\Sigma, A\Omega) \oplus AP$ and $\mathcal{L}(1)$ are incomparable.*

Corollary 10. *Neither $(A\Sigma, A\Omega) \oplus AP$ nor $\mathcal{L}(1)$ is the weakest failure detector for solving consensus in an anonymous asynchronous system.*

Note that it is not yet known whether every problem has exactly one class of weakest failure detectors also in *anonymous* systems, as it is the case for non-anonymous systems (cf. [41]). Therefore, $(A\Sigma, A\Omega) \oplus AP \oplus \mathcal{L}(1)$ could be seen as a promising candidate for a weakest failure detector for consensus in anonymous systems.

7 Conclusions

We introduced two novel message passing models $\mathcal{M}^{\mathrm{anti}(n-k)}$ and $\mathcal{M}^{\mathrm{sink}(n-k)}$ that provide enough synchrony for solving k-set agreement and showed how to implement our generalized $(n-k)$-loneliness failure detector $\mathcal{L}(k)$ in these models. Part of our future research will focus on the still ongoing chase for the weakest failure detector for message passing k-set agreement, both in non-anonymous and anonymous systems.

Acknowledgments

A preliminary version of this paper [10] was accepted at OPODIS '09; some material of the present paper has been relegated to an online supplement [12]. Our work was supported by the Austrian BM:vit FIT-IT project *TRAFT* (proj. no. 812205) and the Austrian Science Foundation (FWF) projects P20529 (PSRTS) and S11405 (RiSE). Peter Robinson was also supported in part by Nanyang Technological University grant M58110000 and Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 grant MOE2010-T2-2-082.

References

[1] M.K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Partial Synchrony Based on Set Timeliness," *Proc. 28th ACM Symp. Principles of Distributed Computing (PODC '09)*, pp. 102-110, 2009.

[2] M.K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Partial Synchrony Based on Set Timeliness," *Distributed Computing*, vol. 25, no. 3, pp. 249-260, 2012.

[3] M.K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "On Implementing Omega with Weak Reliability and Synchrony Assumptions," *Proc. 22nd ACM Symp. Principles of Distributed Computing*, pp. 306-314, July 2003.

[4] M.K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Communication-Efficient Leader Election and Consensus with Limited Link Synchrony," *Proc. 23th ACM Symp. Principles of Distributed Computing (PODC '04)*, pp. 328-337, 2004.

[5] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers, "Of Choices, Failures and Asynchrony: The Many Faces of Set Agreement," *Algorithmica*, vol. 62, no. 1/2, pp. 595-629, 2012.

[6] D. Angluin, "Local and Global Properties in Networks of Processors (Extended Abstract)," *Proc. 12th Ann. ACM Symp. Theory of Computing*, pp. 82-93, 1980.

[7] S. Arevalo, A. Fernandez Anta, D. Imbs, E. Jimenez, and M. Raynal, "Failure Detectors in Homonymous Distributed Systems (with an Application to Consensus)," *Proc. 32nd Int'l IEEE Conf. Distributed Computing Systems (ICDCS '12)*, pp. 275-284, June 2012.

[8] H. Attiya, M. Snir, and M.K. Warmuth, "Computing on an Anonymous Ring," *J. ACM*, vol. 35, no. 4, pp. 845-875, 1988.

[9] M. Biely, M. Hutle, L.D. Penso, and J. Widder, "Relating Stabilizing Timing Assumptions to Stabilizing Failure Detectors Regarding Solvability and Efficiency," *Proc. Ninth Int'l Symp. Stabilization, Safety, and Security of Distributed Systems*, vol. 4838, pp. 4-20, Nov. 2007.

[10] M. Biely, P. Robinson, and U. Schmid, "Weak Synchrony Models and Failure Detectors for Message Passing k-Set Agreement," *Proc. Int'l Conf. Principles of Distributed Systems (OPODIS '09)*, pp. 285-299, Dec. 2009.

[11] M. Biely, P. Robinson, and U. Schmid, "Easy Impossibility Proofs for K-Set Agreement in Message Passing Systems," *Proc. 15th Int'l Conf. Principles of Distributed Systems (OPODIS '11)*, pp. 299-312, 2011.

[12] M. Biely, P. Robinson, and U. Schmid, *Supplement: The Generalized Loneliness Detector and Weak System Models for k-Set Agreement*, 2013.

[13] M. Biely and J. Widder, "Optimal Message-Driven Implementations of Omega with Mute Processes," *ACM Trans. Autonomous and Adaptive Systems*, vol. 4, no. 1, article 4, 2009.

[14] F. Bonnet and M. Raynal, "Looking for the Weakest Failure Detector for K-Set Agreement in Message-Passing Systems: Is Π_k the End of the Road?" *Proc. 11th Int'l Symp. Stabilization, Safety, and Security of Distributed Systems (SSS '09)*, vol. 5873, pp. 129-164, 2009.

[15] F. Bonnet and M. Raynal, "Anonymous Asynchronous Systems: The Case of Failure Detectors," *Proc. 24th Int'l Conf. Distributed Computing*, pp. 206-220, 2010.

[16] F. Bonnet and M. Raynal, "On the Road to the Weakest Failure Detector for K-Set Agreement in Message-Passing Systems," *Theoretical Computer Science*, vol. 412, no. 33, pp. 4273-4284, 2011.

[17] E. Borowsky and E. Gafni, "Generalized FLP Impossibility Result for T-Resilient Asynchronous Computations," *Proc. 25th Ann. ACM Symp. Theory of Computing (STOC '93)*, pp. 91-100, 1993.

[18] Z. Bouzid and C. Travers, "$(\text{anti-}\Omega^x \times \Sigma_z)$-Based k-Set Agreement Algorithms," *Proc. 14th Int'l Conf. Principles of Distributed Systems*, pp. 189-204, 2010.

[19] Z. Bouzid and C. Travers, "Brief Announcement: Anonymity, Failures, Detectors and Consensus," *Proc. 26th Int'l Symp. Distributed Computing (DISC '12)*, pp. 427-428, 2012.

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

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

[22] B. Charron-Bost and A. Schiper, "Uniform Consensus is Harder than Consensus.," *J. Algorithms*, vol. 51, no. 1, pp. 15-37, 2004.

[23] S. Chaudhuri, "More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems," *Information and Computation*, vol. 105, no. 1, pp. 132-158, 1993.

[24] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, "A Realistic Look at Failure Detectors," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '02)*, pp. 345-353, 2002.

[25] C. Delporte-Gallet, "Tight Failure Detection Bounds on Atomic Object Implementations," *J. ACM*, vol. 57, pp. 22:1-22:32, May 2010.

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

[27] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, A.-M. Kermarrec, E. Ruppert, and H. Tran-The, "Byzantine Agreement with Homonyms," *Proc. 30th Ann. SIGACT-SIGOPS Symp. Principles of Distributed Computing (PODC)*, pp. 21-30, 2011.

[28] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and A. Tielmann, "The Weakest Failure Detector for Message Passing Set-Agreement," *Proc. 22nd Int'l Symp. Distributed Computing (DISC '08)*, pp. 109-120, 2008.

[29] D. Dolev, C. Dwork, and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *J. ACM*, vol. 34, no. 1, pp. 77-97, Jan. 1987.

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

[31] A. Fernández and M. Raynal, "From an Asynchronous Intermittent Rotating Star to an Eventual Leader," *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1290-1303, Sept. 2010.

[32] A.F. Anta, S. Rajsbaum, and C. Travers, "Brief Announcement: Weakest Failure Detectors via an Egg-Laying Simulation," *Proc. 28th ACM Symp. Principles of Distributed Computing (PODC)*, pp. 290-291, 2009.

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

[34] M. Függer and U. Schmid, "Reconciling Fault-Tolerant Distributed Computing and Systems-On-Chip," *Distributed Computing*, vol. 24, no. 6, pp. 323-355, 2012.

[35] E. Gafni and P. Kuznetsov, "The Weakest Failure Detector for Solving K-Set Agreement," *Proc. 28th ACM SIGACT-SIGOPS Symp. Principles of Distributed Computing (PODC '09)*, 2009.

[36] R. Guerraoui, M. Herlihy, P. Kouznetsov, N. Lynch, and C. Newport, "On the Weakest Failure Detector Ever," *Proc. 26th Ann. ACM Symp. Principles of Distributed Computing (PODC '07)*, pp. 235-243, Aug. 2007.

[37] R. Guerraoui and A. Schiper, "'Gamma-Accurate' Failure Detectors," *Proc. 10th Int'l Workshop Distributed Algorithms (WDAG '96)*, pp. 269-286, 1996.

[38] M. Herlihy, "Wait-Free Synchronization," *ACM Trans. Programming Language Systems*, vol. 13, no. 1, pp. 124-149, 1991.

[39] M. Herlihy and N. Shavit, "The Asynchronous Computability Theorem for T-Resilient Tasks," *Proc. 25th Ann. ACM Symp. Theory of Computing*, pp. 111-120, 1993.

[40] M. Hutle, D. Malkhi, U. Schmid, and L. Zhou, "Chasing the Weakest System Model for Implementing Omega and Consensus," *IEEE Trans. Dependable and Secure Computing*, vol. 6, no. 4, pp. 269-281, Oct.-Dec. 2009.

[41] P. Jayanti and S. Toueg, "Every Problem Has a Weakest Failure Detector," *Proc. 27th ACM Symp. Principles of Distributed Computing (PODC)*, pp. 75-84, 2008.

[42] M. Larrea, A. Fernández, and S. Arévalo, "Optimal Implementation of the Weakest Failure Detector for Solving Consensus," *Proc. 19th IEEE Symp. Reliable Distributed Systems (SRDS)*, pp. 52-59, Oct. 2000.

[43] D. Malkhi, F. Oprea, and L. Zhou, "Ω Meets Paxos: Leader Election and Stability without Eventual Timely Links," *Proc. 19th Symp. Distributed Computing (DISC '05)*, vol. 3724, pp. 199-213, 2005.

[44] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous Implementation of Failure Detectors," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '03)*, pp. 22-25, June 2003.

[45] A. Mostéfaoui and M. Raynal, "Unreliable Failure Detectors with Limited Scope Accuracy and an Application to Consensus," *Proc. 19th Conf. Foundations of Software Technology and Theoretical Computer Science*, pp. 329-340, 1999.

[46] A. Mostéfaoui and M. Raynal, "K-Set Agreement with Limited Accuracy Failure Detectors," *Proc. 19th Ann. ACM Symp. Principles of Distributed Computing (PODC '00)*, pp. 143-152, 2000.

[47] A. Mostéfaoui, M. Raynal, and J. Stainer, "Relations Linking Failure Detectors Associated with K-Set Agreement in Message-Passing Systems," *Proc. 13th Int'l Conf. Stabilization, Safety, and Security of Distributed Systems (SSS)*, pp. 341-355, 2011.

[48] A. Mostefaoui, M. Raynal, and C. Travers, "Crash-Resilient Time-Free Eventual Leadership," *Proc. 23rd IEEE Symp. Reliable Distributed Systems (SRDS '04)*, pp. 208-217, 2004.

[49] G. Neiger, "Failure Detectors and the Wait-Free Hierarchy (Extended Abstract)," *Proc. 14th Ann. ACM Symp. Principles of distributed computing*, pp. 100-109, 1995.

[50] S.M. Pike, S. Sastry, and J.L. Welch, "Failure Detectors Encapsulate Fairness," *Distributed Computing*, vol. 25, no. 4, pp. 313-333, 2012.

[51] P. Robinson, *Weak System Models for Fault-Tolerant Distributed Agreement Problems* Ph.D. dissertation, Technische Universität Wien, 2011.

[52] P. Robinson and U. Schmid, "The Asynchronous Bounded-Cycle Model," *Proc. 10th Int'l Symp. Stabilization, Safety, and Security of Distributed Systems (SSS'08)*, vol. 5340, pp. 246-262, Nov. 2008.

[53] M. Saks and F. Zaharoglou, "Wait-Free K-Set Agreement Is Impossible: The Topology of Public Knowledge," *SIAM J. Computing*, vol. 29, no. 5, pp. 1449-1483, 2000.

[54] J. Widder and U. Schmid, "The Theta-Model: Achieving Synchrony without Clocks," *Distributed Computing*, vol. 22, no. 1, pp. 29-47, Apr. 2009.

[55] P. Zielinski, "Automatic Classification of Eventual Failure Detectors," *Proc. 21st Int'l Symp. Distributed Computing, (DISC '07)*, vol. 4731, pp. 465-479, Sept. 2007.

[56] P. Zielinski, "Anti-Ω: The Weakest Failure Detector for Set Agreement," *Proc. 27th ACM Symp. Principles of Distributed Computing (PODC '08)*, pp. 55-64, 2008.

Martin Biely received the master's degree in computer science in 2002 and the PhD degree in 2009, both from the Vienna University of Technology, Austria. From 2004 to 2009, he was at the Department of Computer Engineering at the Vienna University of Technology. In 2009 and 2010, he was a Lix-Qualcomm-fellow at Ecole Polytechnique, France. Since 2010, he has been a postdoctoral researcher at the Distributed Systems Laboratory of EPFL, Switzerland. His current research encompasses theoretical, algorithmic, and systems aspects of fault-tolerant distributed computations.

Peter Robinson received the PhD degree in computer science from the Vienna University of Technology. He is a postdoctoral research fellow at the Nanyang Technological University. His current research focuses on theoretical aspects of distributed computing. This comprises the design of distributed algorithms and the development of new models for distributed computation. He received the Best Paper award at the 14th International Conference on Distributed Computing and Networking (ICDCN 2013) and at the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2008).

Ulrich Schmid studied computer science and mathematics and also spent several years in industrial electronics and embedded systems design. He is a full professor and the head of the Embedded Computing Systems Group at the Institut für Technische Informatik at TU Vienna. He authored and coauthored numerous papers in the field of theoretical and technical computer science and received several awards and prices, like the Austrian START-price 1996. His current research interests focus on the mathematical analysis of fault-tolerant distributed algorithms and real-time systems, with special emphasis on their application in systems-on-chips and networked embedded systems.