Distrib. Comput. (2012) 25:249–260
DOI 10.1007/s00446-012-0158-8

Partial synchrony based on set timeliness

Marcos K. Aguilera · Carole Delporte-Gallet · Hugues Fauconnier · Sam Toueg

Received: 22 January 2010 / Accepted: 13 January 2012 / Published online: 16 February 2012

Abstract We introduce a new model of partial synchrony for read-write shared memory systems. This model is based on the simple notion of *set timeliness*—a natural generalization of the seminal concept of *timeliness* in the partially synchrony model of Dwork et al. (J. ACM 35(2):288–323, 1988). Despite its simplicity, the concept of set timeliness is powerful enough to define a family of partially synchronous systems that closely match individual instances of the t-resilient k-set agreement problem among n processes, henceforth denoted (t, k, n)-agreement. In particular, we use it to give a partially synchronous system that is synchronous enough for solving (t, k, n)-agreement, but not enough for solving two incrementally stronger problems, namely, $(t + 1, k, n)$-agreement, which has a slightly stronger resiliency requirement, and $(t, k - 1, n)$-agreement, which has a slightly stronger agreement requirement. This is the first partially synchronous system that separates these sub-consensus problems. The above results show that set timeliness can be used to study and compare the partial synchrony requirements of problems that are strictly weaker than consensus.

A preliminary version of this paper appeared in [3].

M. K. Aguilera (✉)
Microsoft Research Silicon Valley, Mountain View, CA, USA
e-mail: marcos_aguilera_msrsvc@live.com

C. Delporte-Gallet · H. Fauconnier
Université Paris-Diderot, Paris, France

S. Toueg
University of Toronto, Toronto, ON, Canada

1 Introduction

The concept of partial synchrony introduced in the seminal work of Dwork, Lynch and Stockmeyer for message-passing systems is based on the notion of *timeliness*, that is, on the existence of upper bounds on message delays and relative process speeds between every pair of processes [12]. In particular, the upper bound Φ on relative process speeds was defined as follows: "in any contiguous interval containing Φ real-time steps, every correct process must take at least one step. This implies no correct process can run more than Φ times slower than another." In [12], Dwork, Lynch and Stockmeyer showed that consensus can be solved in systems that are timely, even if the timeliness bounds are not known or they hold only eventually.

To define partially synchronous systems that are weaker than those in [12], but are still strong enough to solve consensus, the above notion of timeliness was later refined by considering each pair of processes *individually* [1,2]. In particular, in shared-memory systems, one can define the concept of *process timeliness*, which compares the speed of a *single* process p to the speed of another process q, as follows: *p is timely with respect to q* if, for some integer i, every interval that contains i steps of q contains at least one step of p [4].

The concept of process timeliness, however, seems too strong to study problems that are weaker than consensus, such as *set agreement*. This is because the existence of a single process p that is timely with respect to another process q is sufficient to solve consensus in read-write shared memory systems where at most one process may crash.[1] In fact, all the partially synchronous systems that were previously proposed for message-passing and read-write shared-memory

[1] This follows from a result in [4]; an analogous result for message-passing systems was shown in [1].

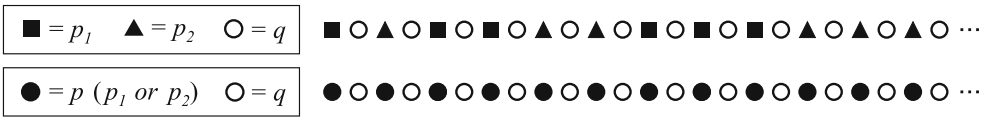

Fig. 1 Example of set timeliness. *Top* shows a schedule with three processes, p_1, p_2, q, in which neither p_1 nor p_2 is timely with respect to q. *Bottom* shows the same schedule where p_1 and p_2 are considered as a single virtual process p, and p is timely with respect to q

are strong enough to solve consensus (under some condition on the number of processes that may crash).

In this paper, we propose a simple generalization of process timeliness, called *set timeliness*, and show that it can be used to study and compare the partial synchrony requirements of problems that are weaker than consensus. Intuitively, this generalization is obtained by considering a set of processes P in the system as a *single entity*, that is, as a "virtual process" p that takes a step whenever any process in P takes a step, and then use the definition of process timeliness on such virtual processes. So, a *set* of processes P is timely with respect to another *set* of processes Q if, for some integer i, every interval that contains i steps of processes in Q contains at least one step of some process in P. As we will see below, the processes in P may not be *individually* timely (i.e., the speed of each process in P may fluctuate beyond any bound), but when they are viewed as a single (cooperating) process they may be timely. So a set of processes may be able to overcome the speed fluctuations of individual members of the set, by working together as a timely virtual process.

A simple example, depicted in Fig. 1, illustrates the definition of set timeliness. Consider the synchrony of processes p_1 and p_2 with respect to process q in schedule $S = [(p_1 \cdot q)^i \cdot (p_2 \cdot q)^i]_{i=1}^{\infty}$. Note that p_1 is *not* timely with respect to q in S, because there are longer and longer sequences of consecutive steps in S where q takes more and more steps while p_1 takes no step at all: intuitively, there are longer and longer periods where p_1 is very slow with respect to q. Similarly, p_2 is *not* timely with respect to q in S. But if we consider p_1 and p_2 as a single virtual process p, then the above schedule S now becomes $(p \cdot q)^{\infty}$, and the virtual process p is indeed timely with respect to q. In other words, if p_1 and p_2 are considered as a single entity (a set of two cooperating processes), then together they are timely with respect to q. In our model of partial synchrony, we say that the set of processes $\{p_1, p_2\}$ is timely with respect to the set $\{q\}$. Similarly, a set of processes $\{p_1, p_2\}$ is timely with respect to a set $\{q_1, q_2, q_3\}$ if, when we remove all the indices from these processes, the resulting virtual process p is timely with respect to virtual process q.

In this paper, we show that set timeliness can be used to study the synchrony requirements of sub-consensus tasks. In particular, we use it to define a family of partially synchronous systems, and prove tight possibility and impossibility results for solving the *t-resilient k-set agreement problem*—a well-known generalization of the wait-free consensus problem [10]—in these systems.[2] More precisely, we do the following:

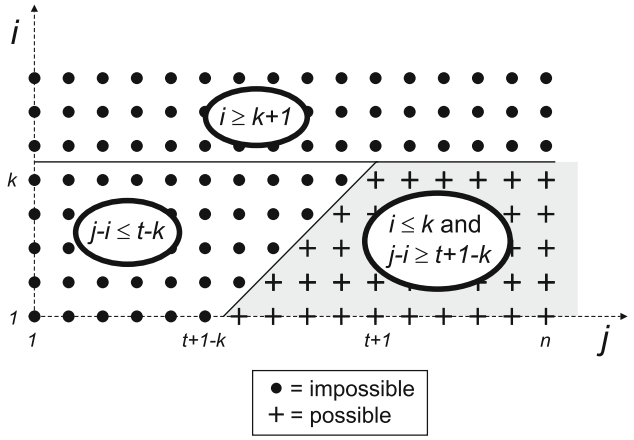

Fig. 2 Solvability of (t, k, n)-agreement in partially synchronous system $\mathcal{S}^i_{j,n}$

1. We define a family of partially synchronous systems, denoted $\mathcal{S}^i_{j,n}$, as follows: $\mathcal{S}^i_{j,n}$ is a read-write shared memory system of n processes where, in every run, at least one set of processes of size i is timely with respect to a set of processes of size j. The family of partially synchronous systems consists of all $\mathcal{S}^i_{j,n}$ where each of i and j ranges from 1 to n.
2. We solve the following general question: *For every possible values of t, k, n, and every possible values of i and j, is the (t, k, n)-agreement problem solvable in partially synchronous system $\mathcal{S}^i_{j,n}$?* The answer to this question is surprisingly simple: (t, k, n)-agreement is solvable in $\mathcal{S}^i_{j,n}$ *if and only if* $i \leq k$ and $j - i \geq (t + 1) - k$.

The above result, illustrated in Fig. 2, gives the first partially synchronous system that separates the (t, k, n)-agreement problem from the following two incrementally

[2] Intuitively, with the t-resilient k-set agreement problem for n processes, henceforth denoted (t, k, n)-*agreement*, there are n processes that propose values, and if at most t of them crash, then each non-faulty process must decide on a proposed value such that there are at most k different decision values. The problem parameter t ranges from 1 (which corresponds to tolerating a single failure) to $n - 1$ (which corresponds to wait-freedom), and parameter k ranges from 1 (which corresponds to consensus) to $n - 1$ (which corresponds to set agreement).

stronger problems: $(t+1,k,n)$-agreement, which has a slightly stronger resiliency requirement, and $(t,k-1,n)$-agreement, which has a slightly stronger agreement requirement. In fact, the result implies that partially synchronous system $\mathcal{S}^k_{t+1,n}$ is synchronous enough for solving (t,k,n)-agreement, but not enough for solving $(t+1,k,n)$-agreement or $(t,k-1,n)$-agreement. The partially synchronous systems that "closely match" the $(t+1,k,n)$-agreement and $(t,k-1,n)$-agreement problems are $\mathcal{S}^k_{t+2,n}$ and $\mathcal{S}^{k-1}_{t+1,n}$, respectively.

Roadmap. This paper is organized as follows. In Sect. 2, we define the notion of set timeliness and use it to define the partially synchronous system $\mathcal{S}^i_{j,n}$. In Sect. 3, we describe the (t,k,n)-agreement problem. In Sect. 4, we prove that (t,k,n)-agreement is solvable in system $\mathcal{S}^k_{t+1,n}$. In Sect. 5, we determine for which values of t, k, n, i and j, the (t,k,n)-agreement is solvable in system $\mathcal{S}^i_{j,n}$. We conclude the paper in Sect. 6 with a discussion of related work.

2 Model

We consider a shared-memory system with n processes $\Pi_n = \{1,\ldots,n\}$, which can communicate with each other via an infinite set Ξ of shared atomic registers.

A *schedule S (in Π_n)* is a finite or infinite sequence of processes (in Π_n). A *step of a schedule* is an element of S. Given a finite schedule S and a schedule S', we denote by $S \cdot S'$ the concatenation of S and S'. Given an infinite schedule S, a process p is *correct in S* if there are infinitely many occurrences of p in S, and p is *faulty in S* otherwise (in this case, we also say that p *crashes in S*).

2.1 Set timeliness

In what follows, P, P', Q, and Q' are sets of processes (subsets of Π_n) and S is a schedule (in Π_n).

Definition 1 *P is timely with respect to Q in S* if there is an integer k such that every sequence of consecutive steps of S that contains k occurrences of processes in Q contains a process in P.

The following observations follow directly from the above definition:

Observation 1 *If P is timely with respect to Q in S, and P' is timely with respect to Q' in S, then $P \cup P'$ is timely with respect to $Q \cup Q'$ in S.*

Observation 2 *If P is timely with respect to Q in S, and $P \subseteq P'$ and $Q' \subseteq Q$, then P' is timely with respect to Q' in S.*

The definition of *set* timeliness given above (Definition 1) is a generalization of the definition of *process* timeliness given in [4]. In fact, Definition 1 can be used to define process timeliness: A process *p is timely with respect to a process q in S* if set $\{p\}$ is timely with respect to set $\{q\}$ in S.

2.2 Systems and partially synchronous systems

A system may be defined by some properties, such as timeliness properties, of its schedules. So we define a system $\mathcal{S}$ as a tuple $\mathcal{S} = (\Pi_n, \Xi, \mathit{Scheds})$ where *Scheds* is a set of schedules in Π_n; intuitively, *Scheds* is the set of schedules that are possible in system $\mathcal{S}$.

The *asynchronous* system of n processes, denoted $\mathcal{S}_n$, is the system $(\Pi_n, \Xi, \mathit{Scheds})$ where *Scheds* is the set of *all* the schedules in Π_n. We define the following family of *partially synchronous systems*: for i and j in $\{1,2,\ldots,n\}$, $\mathcal{S}^i_{j,n}$ is the system of n processes where *at least one* set of processes of size i is timely with respect to *at least one* set of processes of size j. More precisely, let $\mathit{Scheds}^i_{j,n}$ be the set of all the schedules S in Π_n such that in S at least one set of processes of size i is timely with respect to at least one set of processes of size j. We define $\mathcal{S}^i_{j,n} = (\Pi_n, \Xi, \mathit{Scheds}^i_{j,n})$.

We say that a *system $\mathcal{S}'$ is contained in system $\mathcal{S}$*, and write $\mathcal{S}' \subseteq \mathcal{S}$, if every schedule of $\mathcal{S}'$ is also a schedule of $\mathcal{S}$, that is, if $\mathcal{S} = (\Pi_n, \Xi, \mathit{Scheds})$ and $\mathcal{S}' = (\Pi_n, \Xi, \mathit{Scheds}')$, we have $\mathit{Scheds}' \subseteq \mathit{Scheds}$.

Observation 2 implies the following:

Observation 3 *For all i, j, i', and j' in $\{1,2,\ldots,n\}$ such that $i' \leq i$ and $j \leq j'$, $\mathcal{S}^{i'}_{j',n} \subseteq \mathcal{S}^i_{j,n}$.*

Consider the asynchronous system of n processes $\mathcal{S}_n = (\Pi_n, \Xi, \mathit{Scheds})$ and let $1 \leq j \leq i \leq n$. Note that in *every* schedule $S \in \mathit{Scheds}$ of $\mathcal{S}_n$, each set P of i processes is timely with respect to every subset $P' \subseteq P$ of size $j \leq i$, and so $S \in \mathit{Scheds}^i_{j,n}$. Thus $\mathcal{S}_n \subseteq \mathcal{S}^i_{j,n}$. Since $\mathcal{S}^i_{j,n} \subseteq \mathcal{S}_n$ also holds, we have the following:

Observation 4 *For all i and j in $\{1,2,\ldots,n\}$ such that $j \leq i$, $\mathcal{S}^i_{j,n} = \mathcal{S}_n$.*

From the above observation, it suffices to focus on the partially synchronous systems $\mathcal{S}^i_{j,n}$ where $i \leq j$.

2.3 Algorithms and runs

An algorithm $\mathcal{A}$ in a system $\mathcal{S} = (\Pi_n, \Xi, \mathit{Scheds})$ consists of a set of n (infinite or finite) deterministic automata $\mathcal{A}_1,\ldots,\mathcal{A}_n$. By abuse of notation, we identify a process with its automaton. Each process executes by taking steps. In each step, a process p can read or write a shared register and change state (according to p's state transition function in $\mathcal{A}_p$). A *configuration* of $\mathcal{A}$ in $\mathcal{S}$ indicates the state of each process and register. A *run* R of $\mathcal{A}$ in $\mathcal{S}$ is a tuple $R = (I, S, \mathcal{A})$ where I is an initial configuration of $\mathcal{A}$ in $\mathcal{S}$, and S is a schedule in *Scheds*.

3 t-resilient k-set agreement for n processes

The *t-resilient k-set agreement for n processes* problem, denoted (t, k, n)-*agreement*, is defined for $1 \leq t \leq n-1$ and $1 \leq k \leq n$, as follows. Each process in Π_n has an initial value and must decide a value such that the following holds:

- *(Uniform k-agreement)* Processes decide at most k distinct values;
- *(Uniform validity)* If some process decides v then v is the initial value of some process; and
- *(Termination)* If at most t processes are faulty then every correct process eventually decides some value.

Note that $(t, n-1, n)$-agreement is also called *t-resilient set agreement*, and $(t, 1, n)$-agreement is also called *t-resilient* consensus. When $t = n-1$, we get the *wait-free* versions of these problems, which are simply called *set agreement* and *consensus*, respectively.

Observation 5 *For all $1 \leq t \leq n-1$ and $1 \leq k \leq n$, if (t, k, n)-agreement can be solved in a system $\mathcal{S}$ then it can also be solved in every system $\mathcal{S}'$ such that $\mathcal{S}' \subseteq \mathcal{S}$.*

Observations 3 and 5 imply the following:

Observation 6 *For all $1 \leq t \leq n-1$ and $1 \leq k \leq n$, if (t, k, n)-agreement can be solved in a system $\mathcal{S}^i_{j,n}$, where $1 \leq i \leq j \leq n$, then it can also be solved in every system $\mathcal{S}^{i'}_{j',n}$ such that $1 \leq i' \leq i$ and $j \leq j' \leq n$.*

4 Solving t-resilient k-set agreement for n processes in system $\mathcal{S}^k_{t+1,n}$

To show that t-resilient k-set agreement for n processes can be solved in $\mathcal{S}^k_{t+1,n}$, we use the t-resilient version of k-anti-Ω—a failure detector given in [14,26]. In the following, we define t-resilient k-anti-Ω, we give an algorithm that implements t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$, and we observe that, from a result in [14], t-resilient k-anti-Ω can be used to solve (t, k, n)-agreement.

4.1 Failure detector k-anti-Ω

Let t and k be such that $1 \leq t \leq n-1$ and $1 \leq k \leq n-1$. With the t-resilient k-anti-Ω failure detector, every process p has a local variable $fdOutput_p$ that holds a set of $n-k$ processes, such that the following property holds: if at most t processes are faulty then there exists a correct process c and a time after which, for every correct process p, c is not in $fdOutput_p$. Note that when $t = n-1$, t-resilient k-anti-Ω is just the k-anti-Ω failure detector defined in [14,26].[3]

4.2 Algorithm for t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$

We first note that if $1 \leq t < k \leq n-1$ then it is trivial to implement t-resilient k-anti-Ω in the asynchronous system $\mathcal{S}_n$: the implementation algorithm constantly outputs the same, arbitrarily chosen set B of $n-k$ processes at all processes and at all times; since $t < k$, in every run where at most t processes crash there is at least one correct process that is *not* in the set B, and this satisfies the specification of t-resilient k-anti-Ω.

We now consider the case where $1 \leq k \leq t \leq n-1$, and give an algorithm that implements t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$. In other words, this algorithm works in a system where every run has two sets P and Q of sizes k and $t+1$, respectively, such that P is timely with respect to Q. In the following, $\mathbf{\Pi}^k_n$ denotes the set of all subsets of Π_n of size k.

The basic idea of our algorithm is that each process p has a heartbeat that it increments periodically, and process p has a timeout timer on each set A in $\mathbf{\Pi}^k_n$. Process p resets the timer for A whenever it sees that the heartbeat of *any* process in A has increased. If p's timer for A expires (the process times out on A), process p increments the timeout that it subsequently uses for A, and p also increments a shared register $Counter[A, p]$. This shared register represents a "badness" counter for A as seen by process p. Note that $Counter[A, p]$ is monotonically nondecreasing, so either it grows to infinity or it eventually stops changing. We define the accusation counter of a set A to be the $(t+1)$-st smallest value of $Counter[A, *]$. Intuitively, the accusation counter of A has two properties: (1) if at least $n-t$ entries of $Counter[A, *]$ grow to infinity then the accusation counter of A also grows to infinity, and (2) if at least $t+1$ entries of $Counter[A, *]$ eventually stop changing then the accusation counter of A also eventually stops changing. Each process p repeatedly picks the set that has the smallest accusation counter, breaking ties using some arbitrary total order on $\mathbf{\Pi}^k_n$. This set is denoted $winnerset_p$, and p outputs the set $\Pi_n - winnerset_p$ as the output of k-anti-Ω.

The detailed algorithm is shown in Fig. 3. Each process executes an infinite loop, in which the process reads $Counter[A, q]$ for each set A in $\mathbf{\Pi}^k_n$ and each process $q \in \Pi_n$, calculates the accusation counter of each set A, chooses a winner, and sets the output of k-anti-Ω accordingly. The process then increments its heartbeat, checks the heartbeats of each process q and, if the heartbeat has increased, it resets the timers of all the sets in $\mathbf{\Pi}^k_n$ containing q. Finally,

[3] So $(n-1)$-resilient 1-anti-Ω is equivalent to failure detector Ω [9], and $(n-1)$-resilient $(n-1)$-anti-Ω is also called anti-Ω [26].

SHARED REGISTERS:

$\forall p \in \Pi_n$: *Heartbeat*$[p] = 0$
$\forall A \in \mathbf{\Pi}_n^k, \forall q \in \Pi_n$: *Counter*$[A, q] = 0$ // $\mathbf{\Pi}_n^k$ is the set of all subsets of Π_n of size k

CODE OF PROCESS $p \in \Pi_n$:

Local variables:

fdOutput = any set of processes of size $n - k$
winnerset $= \emptyset$
myHb $= 0$
$\forall q \in \Pi_n$: *prevHeartbeat*$[q] = 0$
$\forall A \in \mathbf{\Pi}_n^k$: *timeout*$[A] = 2$
$\forall A \in \mathbf{\Pi}_n^k$: *timer*$[A]$ = *timeout*$[A]$
$\forall A \in \mathbf{\Pi}_n^k$: *accusation*$[A] = 0$
$\forall A \in \mathbf{\Pi}_n^k, \forall q \in \Pi_n$: *cnt*$[A, q] = 0$
hbq $= 0$

Main code:

```
1  repeat forever
       // choose FD output
2      for each ⟨A, q⟩ ∈ Π_n^k × Π_n do cnt[A, q] ← read(Counter[A, q])
3      for each A ∈ Π_n^k do accusation[A] ← (t + 1)-st smallest value of cnt[A, ∗]
4      winnerset ← argmin_{A∈Π_n^k}{(accusation[A], A)}      // break ties using a total order on Π_n^k
5      fdOutput ← Π_n − winnerset
       // bump heartbeat
6      myHb ← myHb + 1
7      write(Heartbeat[p], myHb)
       // check other processes' heartbeat
8      for each q ∈ Π_n do
9          hbq ← read(Heartbeat[q])
10         if hbq > prevHeartbeat[q] then
11             for each A ∈ Π_n^k do
12                 if q ∈ A then timer[A] ← timeout[A]
13             prevHeartbeat[q] ← hbq
       // check for expiration of set timers
14     for each A ∈ Π_n^k do
15         timer[A] ← timer[A] − 1
16         if timer[A] = 0 then
17             timeout[A] ← timeout[A] + 1
18             timer[A] ← timeout[A]
               // increment Counter[A, p] based on the value read in line 2
19             write(Counter[A, p], cnt[A, p] + 1)
```

Fig. 3 Algorithm for t-resilient k-anti-Ω in system $\mathcal{S}_{t+1,n}^k$

process p checks if the timers have expired, and increments *Counter*$[A, p]$ for the sets A whose timer expired.

Intuitively, this algorithm works because there is at least one set P of size k that is timely with respect to some set Q of size $t + 1$. As we shall see, this implies that eventually every process $q \in Q$ stops increasing *Counter*$[P, q]$. So, at least $t + 1$ entries of *Counter*$[P, *]$ eventually stop changing. Thus, the accusation counter of P also eventually stops changing. Among all sets whose accusation counter stops changing, one of them, say A_0, ends up with the smallest accusation counter, and eventually all correct processes pick this set as the winner and output $\Pi_n - A_0$. Note that A_0 must have a correct process: if all processes in A_0 were faulty then all correct processes (there are at least $n - t$ of them) would keep timing out on A_0 and so at least $n - t$ entries of *Counter*$[A_0, *]$ would grow to infinity, so the accusation counter of A_0 would also grow to infinity.

We now sketch a correctness proof. Let k, t, n be such that $1 \leq k \leq t \leq n - 1$. Henceforth, we consider an arbitrary run R of the algorithm of Fig. 3 in system $\mathcal{S}_{t+1,n}^k$. In the proof, the local variable *var* of a process p is denoted by var_p. Let S be the schedule of run R. Henceforth, "steps" refer to steps

in S, and a "correct" or "faulty" process refers to a correct or faulty process in S. If we say that a process crashes, we mean it crashes in S.

We must show that if at most t processes crash then there exists a correct process c and a time after which, for every correct process p, c is not in $fdOutput_p$. Henceforth, suppose that at most t processes crash.

Lemma 1 *Let $A \in \mathbf{\Pi}_n^k$ and suppose that A is timely with respect to some set $B \subseteq \Pi_n$ in S. Then there exists a constant κ such that every sequence of consecutive steps of S containing κ steps of processes in B contains a step of a process in A that writes in line 7.*

Proof Each loop iteration has a bounded number of steps, so the result follows from the definition of what it means for set A to be timely with respect to B in S (Definition 1). □

Note that, for any $A \in \mathbf{\Pi}_n^k$, $Counter[A, q]$ can be modified only by process q, and only by incrementing it. Thus, $Counter[A, q]$ is monotonically nondecreasing and we have the following:

Lemma 2 *For every $A \in \mathbf{\Pi}_n^k$ and every $q \in \Pi_n$, either eventually $Counter[A, q]$ stops changing or it grows monotonically to infinity.*

We now give a sufficient condition for $Counter[A, q]$ to eventually stop changing.

Lemma 3 *For every $A \in \mathbf{\Pi}_n^k$ and every $B \subseteq \Pi_n$, if A is timely with respect to B in S then for every process $b \in B$, there is a time after which $Counter[A, b]$ stops changing.*

Proof From Lemma 1, there exists a constant κ such that, every sequence of consecutive steps of S containing κ steps of processes in B contains a step of a process in A that writes in line 7. In this line, $Heartbeat[a]$ is incremented for some $a \in A$. Therefore, for every process $b \in B$, there exists a constant κ' such that $timer_b[A]$ is reset to $timeout_b[A]$ at least once every κ' steps of b. Thus, since b increases $timeout_b[A]$ each time it finds that $timer_b[A] = 0$, there is a time after which b does not find that $timer_b[A] = 0$ in line 16. So there is a time after which $Counter[A, b]$ stops changing. □

We now give a sufficient condition for $Counter[A, q]$ to grow to infinity.

Lemma 4 *For every $A \in \mathbf{\Pi}_n^k$, if every process in A crashes then for every correct process b, $Counter[A, b]$ grows to infinity.*

Proof If every process in A crashes then eventually no process in A increments its entry in the *Heartbeat* vector. Thus, for every correct process b, there is a time after which b does not set $timer_b[A]$ to $timeout_b[A]$ in line 12. Then b finds that $timer_b[A] = 0$ in line 16 infinitely often, and writes $Counter[A, b]$ infinitely often in line 19. Therefore $Counter[A, b]$ grows to infinity. □

We now define a pseudo-variable $counter(A)$ that depends on the current values of $Counter[A, *]$.

Definition 2 For every $A \in \mathbf{\Pi}_n^k$, $counter(A)$ is the $(t+1)$-st smallest entry of $Counter[A, *]$.

Note that, since $t \leq n-1$, the $(t+1)$-st smallest entry of $Counter[A, *]$ is well-defined. Moreover, since each entry of $Counter[A, *]$ is monotonically nondecreasing, $counter(A)$ is also monotonically nondecreasing. Thus, we can define the following:

Definition 3 For every $A \in \mathbf{\Pi}_n^k$, we define $c(A)$ as follows. If $counter(A)$ grows to infinity then $c(A) = \infty$. Otherwise, $counter(A)$ eventually stops changing and we let $c(A)$ be its final value.

We now establish a relation between $c(A)$ and the entries of $Counter[A, *]$.

Lemma 5 *For every $A \in \mathbf{\Pi}_n^k$, $c(A) = \infty$ if and only if at least $n - t$ entries of $Counter[A, *]$ grow to infinity.*

Proof Let $A \in \mathbf{\Pi}_n^k$. To show the "if" part of the lemma, suppose that at least $n-t$ entries of $Counter[A, *]$ grow to infinity. Then the smallest $t+1$ entries of $Counter[A, *]$ includes at least one entry that grows to infinity. Thus, $counter(A)$ also grows to infinity, so $c(A) = \infty$.

We now show the "only if" part of the lemma, by showing its contrapositive. Suppose that fewer than $n - t$ entries of $Counter[A, *]$ grow to infinity. Then at least $t + 1$ entries of $Counter[A, *]$ eventually stop changing. Thus, eventually the smallest $t + 1$ entries of $Counter[A, *]$ all stop changing (since an entry either stops changing or it grows monotonically to infinity). Thus, $counter(A)$ also eventually stops changing, so $c(A) < \infty$. □

Lemma 6 *For every $A \in \mathbf{\Pi}_n^k$, if A is timely with respect to some set B of size $t + 1$ in S then $c(A) < \infty$.*

Proof By Lemmas 3 and 5. □

Lemma 7 *For every $A \in \mathbf{\Pi}_n^k$, if every process in A crashes then $c(A) = \infty$.*

Proof By Lemmas 4 and 5, and the fact that there are at least $n - t$ correct processes. □

We now define A_0 to be the set of k processes with smallest $c(A)$, breaking ties using a total order on $\mathbf{\Pi}_n^k$.

Definition 4 Let $A_0 = argmin_{A \in \mathbf{\Pi}_n^k}\{(c(A), A)\}$.

Lemma 8 $c(A_0) < \infty$.

Proof Recall that we are considering a run R of the algorithm in system $\mathcal{S}_{t+1,n}^k$. So there must be two sets of processes P and Q of size k and $t + 1$, respectively, such that P is timely with respect to Q in the schedule S of run R. By Lemma 6, $c(P) < \infty$. The result follows since $c(A_0) \leq c(P)$ by definition of A_0. □

Lemma 9 *A_0 has a correct process.*

Proof Immediate from Lemmas 8 and 7. □

We now establish a relation between $c(A)$ and the local variable $accusation_q[A]$ of a correct process q.

Lemma 10 *For every $A \in \mathbf{\Pi}_n^k$ and every correct process q, if $c(A) < \infty$ then there is a time after which $accusation_q[A] = c(A)$; if $c(A) = \infty$ then $accusation_q[A]$ grows to infinity.*

Proof Let $A \in \mathbf{\Pi}_n^k$ and q be a correct process. Since q is correct, for every process p, q sets $cnt_q[A, p]$ to $Counter[A, p]$ in line 2 infinitely often. Thus, for each process p, $cnt_q[A, p]$ eventually stops changing if and only if $Counter[A, p]$ eventually stops changing. Thus, by the way q sets $accusation_q[A]$ in line 3 and by definition of $counter(A)$, we have that $accusation_q[A]$ eventually stops changing if and only if $counter(A)$ eventually stops changing. Moreover, if $counter(A)$ eventually stops changing then its final value $c(A)$ is also the final value of $accusation_q[A]$. The result now follows by the definition of $c(A)$: if $c(A) < \infty$ then $counter(A)$ eventually stops changing and so, by the above, $accusation_q[A]$ also stops changing and their final values are the same; if $c(A) = \infty$ then $counter(A)$ grows to infinity, and so $accusation_q[A]$ also grows to infinity. □

Finally, we show that every correct process outputs $\Pi_n - A_0$.

Lemma 11 *There is a time after which every correct process outputs $\Pi_n - A_0$.*

Proof Let p be any correct process. By Lemma 8, $c(A_0) < \infty$. Thus, by Lemma 10, there is a time after which $accusation_p[A_0] = c(A_0)$.

It is clear that there is a time after which p can pick only A_0 in line 4, because if $A \neq A_0$ then either (a) $c(A) = \infty$, so by Lemma 10 $accusation_p[A]$ grows to infinity, and so there is a time after which $(accusation_p[A], A) > (accusation_p[A_0], A_0)$, or (b) $c(A) < \infty$, so by Lemma 10 and the definition of A_0, there is a time after which $(accusation_p[A], A) = (c(A), A) > (c(A_0), A_0) = (accusation_p[A_0], A_0)$. □

Theorem 7 *For every k, t, n such that $1 \leq k \leq t \leq n-1$, the algorithm in Fig. 3 implements t-resilient k-anti-Ω in system $\mathcal{S}_{t+1,n}^k$.*

Proof Consider any run of the algorithm in Fig. 3 in system $\mathcal{S}_{t+1,n}^k$. Suppose that at most t processes crash. It is clear that the output at each process is a set of $n - k \geq 1$ processes. By Lemma 9, there is a correct process c in A_0. By Lemma 11, there is a time after which every correct process outputs $\Pi_n - A_0$, which does not contain c. Hence all the requirements of t-resilient k-anti-Ω are satisfied. □

It is worth noting that a trivial modification of the algorithm in Fig. 3 implements the t-resilient version of Ω^k [21], a failure detector that is stronger than t-resilient k-anti-Ω, in system $\mathcal{S}_{t+1,n}^k$. Intuitively, t-resilient Ω^k satisfies the following property: if at most t processes are faulty, there is a set A of k processes that includes a correct process such that, eventually, the failure detector outputs A at all correct processes. Consider the trivial modification of the algorithm in Fig. 3 whereby each process outputs the set *winnerset* (instead of $fdOutput \leftarrow \Pi_n - winnerset$) in line 5 of the algorithm's code. From Lemmas 9 and 11, it is clear that this algorithm satisfies the property of t-resilient Ω^k when executed in system $\mathcal{S}_{t+1,n}^k$.

4.3 Using t-resilient k-anti-Ω to solve (t, k, n)-agreement

By Theorem 7, one can implement t-resilient k-anti-Ω in system $\mathcal{S}_{t+1,n}^k$. Moreover, from [26] it is known that t-resilient k-anti-Ω can be used to solve the (t, k, n)-agreement problem in the asynchronous system $\mathcal{S}_n$. So we can implement t-resilient k-anti-Ω and then use it to solve (t, k, n)-agreement problem in system $\mathcal{S}_{t+1,n}^k$. This gives the following:

Theorem 8 *For every t, k, n such that $1 \leq k \leq t \leq n-1$, the (t, k, n)-agreement problem can be solved in system $\mathcal{S}_{t+1,n}^k$.*

When $1 \leq t < k \leq n$ it is trivial to solve (t, k, n)-agreement in the asynchronous system $\mathcal{S}_n$. So we have the following:

Corollary 1 *For every t, k, n such that $1 \leq t \leq n-1$ and $1 \leq k \leq n$, the (t, k, n)-agreement problem can be solved in system $\mathcal{S}_{t+1,n}^k$.* □

5 Determining if (t, k, n)-agreement is solvable in $\mathcal{S}_{j,n}^i$

We now present our main result: for every $1 \leq k \leq t \leq n-1$, and every $1 \leq i \leq j \leq n$, we determine whether the (t, k, n)-agreement problem is solvable or not solvable in the partially synchronous system $\mathcal{S}_{j,n}^i$. To do so, we first consider the special case where $t = k$, and prove the following theorem:

Theorem 9 *For every k and n such that $1 \leq k \leq n-1$,*

1. *The (k, k, n)-agreement problem can be solved in system $\mathcal{S}_{n,n}^k$.*
2. *The (k, k, n)-agreement problem cannot be solved in system $\mathcal{S}_{n,n}^{k+1}$.*

Proof Let k and n be such that $1 \leq k \leq n-1$.

1. By Theorem 8, (k, k, n)-agreement can be solved in $\mathcal{S}^k_{k+1,n}$. Since $k + 1 \leq n$, by Observation 6, (k, k, n)-agreement can also be solved in $\mathcal{S}^k_{n,n}$.
2. Suppose, for contradiction, that there is an algorithm $\mathcal{A}$ that solves (k, k, n)-agreement in the partially synchronous system $\mathcal{S}^{k+1}_{n,n}$. We prove below that this implies that there is an algorithm $\mathcal{A}'$ that solves (k, k, n)-agreement in the *completely asynchronous system* $\mathcal{S}_n$. Since $1 \leq k < n$, this contradicts a well-known impossibility result [5,7] and concludes the proof.

The algorithm $\mathcal{A}'$ that solves (k, k, n)-agreement in system $\mathcal{S}_n$ is shown in Fig. 4. Intuitively, $\mathcal{A}'$ consists of a scheduler that, when executed in system $\mathcal{S}_n$, emulates runs of algorithm $\mathcal{A}$ in the partially synchronous system $\mathcal{S}^{k+1}_{n,n}$. This emulation works as long as at most k out of the n processes crash during the emulation; if more than k processes crash then the emulation of a run of $\mathcal{A}$ in $\mathcal{S}^{k+1}_{n,n}$ "blocks".

Since (a) $\mathcal{A}'$ executed in system $\mathcal{S}_n$ emulates runs of $\mathcal{A}$ in system $\mathcal{S}^{k+1}_{n,n}$ provided that at most k processes crash (and just blocks otherwise), (b) $\mathcal{A}$ executed in system $\mathcal{S}^{k+1}_{n,n}$ solves (k, k, n)-agreement, and (c) the (k, k, n)-agreement problem requires termination only if at most k processes crash, we conclude that $\mathcal{A}'$ solves (k, k, n)-agreement in $\mathcal{S}_n$.

We now explain how the algorithm $\mathcal{A}'$ of Fig. 4 works. To emulate a run of $\mathcal{A}$ in the partially synchronous system $\mathcal{S}^{k+1}_{n,n}$, the n processes of the asynchronous system $\mathcal{S}_n$ schedule the execution of the steps of $\mathcal{A}$ such that every set of $k + 1$ processes is timely with respect to every process. To do so, they ensure that no process can execute too many steps of $\mathcal{A}$ "alone", that is, without at least one step of $\mathcal{A}$ executed by (some process in) every set of size $k + 1$. To enforce this, a process can take its next step of $\mathcal{A}$ if and only if at least $n - k$ processes allow it to do so.

To implement this scheduling policy, we use an $n \times n$ matrix of one-bit read-write shared registers denoted *allow_step*: intuitively, $allow\_step[p, q] = 1$ means that process p is currently allowing process q to take its next step of $\mathcal{A}$. In the algorithm, each process p repeatedly does the following: first p sets $allow\_step[p, q] = 1$ for every process q, that is, p allows every q to take its next step of $\mathcal{A}$; then, p waits until $allow\_step[q, p] = 1$ for at least $n - k$ processes q, that is, until it sees that at least $n - k$ processes allow it to take a step of $\mathcal{A}$; when this occurs, p takes its next step of $\mathcal{A}$, and it resets $allow\_step[q, p]$ to 0 for all processes q.

To prove the correctness of algorithm $\mathcal{A}'$, consider an arbitrary run R of $\mathcal{A}'$ in system $\mathcal{S}_n$ such that at most k processes crash (where $1 \leq k \leq n - 1$). We now show that R emulates some run of algorithm $\mathcal{A}$ in $\mathcal{S}^{k+1}_{n,n}$. To do so, we first prove that algorithm $\mathcal{A}'$ does not block, that is, in run R every correct process takes an infinite number of steps of $\mathcal{A}$. Then we show that every set of size $k + 1$ is timely in the run of $\mathcal{A}$ emulated by R.

CLAIM 1: *Every correct process executes infinitely many iterations of the while loop (lines* 1–6*).*

Suppose, for contradiction, that Claim 1 does not hold, and let τ be the earliest time when some correct process $p \in \Pi_n$ *starts* an iteration of the while loop (lines 1–6) in which it gets stuck. Note that process p must be stuck executing forever in the repeat-until loop of lines 3–4. Thus, from time τ onwards, process p never sets $allow\_step[q, p]$ to 0 (in line 6) for any $q \in \Pi_n$ (*).

By the definition of τ, every correct process q starts at least one iteration of the while loop at some time $\tau_q \geq \tau$. Thus, every correct process q sets $allow\_step[q, p]$ to 1 in line 2 at some time $\tau'_q \geq \tau$. So, from (*), for every correct process q there is a time after which $allow\_step[q, p] = 1$. Since there are at least $n - k$ such correct processes q, this implies that process p does not get stuck forever in the repeat-until loop of lines 3–4, a contradiction showing that Claim 1 holds.

By Claim 1, we immediately have the following:

SHARED REGISTERS:

allow_step, an $n \times n$ matrix of one-bit read-write registers, initially arbitrary

CODE OF PROCESS $p \in \Pi_n$:

Local variables:

temp, a vector in $\{0, 1\}^n$, initially arbitrary

Main code:

```
1  while true do
2      for each q ∈ Π_n do allow_step[p, q] ← 1                    // p allows q to take a step of A
3      repeat for each q ∈ Π_n do temp[q] ← allow_step[q, p]
4      until |{q : q ∈ Π_n ∧ temp[q] = 1}| ≥ n − k                 // wait for n − k processes to allow p to take a step of A
5      process p executes a step of algorithm A
6      for each q ∈ Π_n do allow_step[q, p] ← 0                    // clear allow_step[∗, p]
```

Fig. 4 Algorithm $\mathcal{A}'$ executed in system $\mathcal{S}_n$ emulates a run of algorithm $\mathcal{A}$ in system $\mathcal{S}^{k+1}_{n,n}$

CLAIM 2: *Every correct process executes infinitely many steps of algorithm $\mathcal{A}$.*

The execution of a step of algorithm $\mathcal{A}$ by process p is called an *$\mathcal{A}$-step of p*. An $\mathcal{A}$-step of p *occurs at time* τ if it takes effect at time τ.

We now prove that for every process p and set S of $k+1$ processes, between every two consecutive $\mathcal{A}$-steps of p, some process q in S sets $allow\_step[q, p]$ to 1 in line 2. More precisely, we have the following:

CLAIM 3: *Let p be any process and S be any set of $k+1$ processes. Consider any two consecutive $\mathcal{A}$-steps of p, and let τ_1 and τ_2 be the times when these steps occur. There is a process $q \in S$ that sets $allow\_step[q, p]$ to 1 in line 2 at some time τ_q where $\tau_1 < \tau_q < \tau_2$.*

To show this claim, let p be a process and S be a set of $k+1$ processes. Let τ_1 and τ_2 be the times when two consecutive $\mathcal{A}$-steps of p occur. Note that after time τ_1 and before time τ_2, process p sets $allow\_step[q, p]$ to 0 for every $q \in S$. After doing this, and before time τ_2, process p reads $allow\_step[q, p]$ for every $q \in \Pi_n$ in line 3 of the repeat-until loop, and finds that $allow\_step[q, p] = 1$ for $n-k$ values of q in line 4 of this loop. Since S contains $k+1$ processes, one of these q must be in the set S. So for this $q \in S$, during the interval of time $]\tau_1, \tau_2[$, process p first writes 0 into $allow\_step[q, p]$ and then reads $allow\_step[q, p]$ and gets the value 1. Thus, this $q \in S$ must set $allow\_step[q, p]$ to 1 in line 2 at some time τ_q such that $\tau_1 < \tau_q < \tau_2$, and this shows that Claim 3 holds.

We now prove that for every process p and set S of $k+1$ processes, between every $k+3$ consecutive $\mathcal{A}$-steps of p, some process in S takes an $\mathcal{A}$-step. More precisely, we have the following:

CLAIM 4: *Let p be any process and S be any set of $k+1$ processes. Consider any $k+3$ consecutive $\mathcal{A}$-steps of p, and let τ_1 and τ_2 be the times when the first and the last of these steps occur. There is a process $q \in S$ such that an $\mathcal{A}$-step of q occurs at some time τ_q where $\tau_1 < \tau_q < \tau_2$.*

To prove Claim 4, let p be a process and S be a set of $k+1$ processes. Consider $k+3$ consecutive $\mathcal{A}$-steps of p, and let τ_1 and τ_2 be the times when the first and the last of these steps occur. By applying Claim 3 $k+2$ times, once for every two consecutive $\mathcal{A}$-steps of p that occur during the interval of time $[\tau_1, \tau_2]$, we deduce that processes in S execute line 2 at least $k+2$ times during the interval $]\tau_1, \tau_2[$. Since S has $k+1$ processes, some process q in S executes line 2 at least twice during interval $]\tau_1, \tau_2[$, and hence q executes a step of $\mathcal{A}$ during that interval. So an $\mathcal{A}$-step of $q \in S$ occurs at some time τ_q where $\tau_1 < \tau_q < \tau_2$, and hence Claim 4 holds.

Claim 4 immediately implies that *every set S of $k+1$ processes is timely* in the (schedule of the) emulated run of $\mathcal{A}$. So this emulated run is a run of $\mathcal{A}$ in system $\mathcal{S}^{k+1}_{n,n}$.

The above holds under our initial assumption that at most k processes crash in the run R of $\mathcal{A}'$ in $\mathcal{S}_n$. It is easy to see that if more than k processes crash, the emulation of a run of $\mathcal{A}$ by $\mathcal{A}'$ proceeds correctly until it blocks, and so $\mathcal{A}'$ never violates the safety properties of (k, k, n)-agreement, namely, uniform k-agreement and uniform validity.

Since $\mathcal{A}$ solves (k, k, n)-agreement in system $\mathcal{S}^{k+1}_{n,n}$, we conclude that $\mathcal{A}'$ solves (k, k, n)-agreement in system $\mathcal{S}_n$. □

Note that, in the above proof, we consider a system with up to k crashes and we show how it can emulate a system where *every* set of $k+1$ processes is timely with respect to every process. To prove Theorem 9, however, it suffices to emulate a weaker system where *some* set of $k+1$ processes is timely with respect to every process, as required by $\mathcal{S}^{k+1}_{n,n}$. To obtain this weaker emulation, we can simplify Fig. 4 as follows: pick any fixed set A of $k+1$ processes (e.g., $A = \{1, \ldots, k+1\}$) and allow each process to take its next step if and only if at least some process in A allows it to do so. This emulation ensures that A is timely with respect to every process. Moreover, since at most k processes may crash, it also ensures that correct processes take infinitely many steps. In the proof of Theorem 9, we give the stronger emulation because it suggests a possibly interesting and stronger partially synchronous model to study, where *every* set of some fixed size is timely.

We now state and prove the main result of the paper:

Theorem 10 *For every t, k, n such that $1 \le k \le t \le n-1$ and every i and j such that $1 \le i \le j \le n$, the (t, k, n)-agreement problem can be solved in system $\mathcal{S}^i_{j,n}$ if and only if $i \le k$ and $j - i \ge t + 1 - k$.*

Proof Let $1 \le k \le t \le n-1$ and $1 \le i \le j \le n$.

1. Suppose $i \le k$ and $j - i \ge t + 1 - k$. *We show that (t, k, n)-agreement can be solved in $\mathcal{S}^i_{j,n}$.*
   We consider two cases:
   (a) $j \ge t+1$. By Theorem 8, (t, k, n)-agreement can be solved in system $\mathcal{S}^k_{t+1,n}$. Since $i \le k$ and $j \ge t+1$, by Observation 6, (t, k, n)-agreement can be also solved in system $\mathcal{S}^i_{j,n}$.
   (b) $j < t+1$. Let S be an arbitrary schedule of system $\mathcal{S}^i_{j,n}$. By definition, in S there is a set of processes P_i of size i that is timely with respect to a set of processes P_j of size j. Since $n \ge t+1$, we have $n - j \ge t + 1 - j$. So, among the n processes in Π_n, there are at least $t+1-j$ processes that are *not* in the set P_j. Let Q be a set of $t+1-j$ processes that are not in P_j (since $j < t+1$, this set is not empty).
   Let $P_{t+1} = P_j \cup Q$ and $P_l = P_i \cup Q$. Since P_j and Q are disjoint, the size of P_{t+1} is $j + (t+1-j) = t+1$. P_i and Q are not necessarily disjoint, so the size of P_l is l such that $i \le l \le i + (t+1-j) \le t+1$.

Since P_i is timely with respect to P_j in schedule S, and Q is timely with respect to itself in S, by Observation 1, $P_l = P_i \cup Q$ is timely with respect to $P_{t+1} = P_j \cup Q$ in S. Thus, since $|P_l| = l$ and $|P_{t+1}| = t+1$, every schedule S of $\mathcal{S}^i_{j,n}$ is also a schedule of $\mathcal{S}^l_{t+1,n}$. Hence $\mathcal{S}^i_{j,n} \subseteq \mathcal{S}^l_{t+1,n}$.
By Corollary 1, (t, l, n)-agreement can be solved in $\mathcal{S}^l_{t+1,n}$. Since $\mathcal{S}^i_{j,n} \subseteq \mathcal{S}^l_{t+1,n}$, by Observation 5, (t, l, n)-agreement can also be solved in $\mathcal{S}^i_{j,n}$. By assumption, $j - i \geq t + 1 - k$, so $k \geq t + 1 + i - j$ and therefore $k \geq l$. So (t, k, n)-agreement can be solved in $\mathcal{S}^i_{j,n}$.

2. Suppose $i > k$ or $j - i < t + 1 - k$. *We show that* (t, k, n)*-agreement cannot be solved in* $\mathcal{S}^i_{j,n}$. We consider two cases:

   (a) $i > k$. By Theorem 9 part (2), (k, k, n)-agreement cannot be solved in system $\mathcal{S}^{k+1}_{n,n}$. Since $i \geq k + 1$ and $j \leq n$, by Observation 6, (k, k, n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$. Since $k \leq t$, (t, k, n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$.

   (b) $i \leq k$. Then, by our hypothesis, we must have $j - i < t + 1 - k$, and so $1 \leq i \leq k < t + 1 - (j - i)$. We claim that (t, k, n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$. Suppose, for contradiction, that (t, k, n)-agreement can be solved in $\mathcal{S}^i_{j,n}$. We now prove that this implies that, for some $1 \leq \ell < m$, (ℓ, ℓ, m)-agreement can be solved in the asynchronous system $\mathcal{S}_m$—a contradiction to a well-known impossibility result [5,7].
   Let $\ell = t - (j - i)$ and $m = n - (j - i)$. Since $1 < t + 1 - (j - i)$ and $n > t$, we have $1 \leq \ell < m$. Consider the asynchronous system $\mathcal{S}_m$. The $m \geq 2$ processes of this system can solve (ℓ, ℓ, m)-agreement as follows. They pretend they are in a larger system $\mathcal{S}$ with $m + (j - i)$ processes, where the additional $(j - i)$ fictitious processes never take a step. Intuitively, in system $\mathcal{S}$, the $(j - i)$ fictitious processes are crashed from the start. Note that the simulated system $\mathcal{S}$ has a total of $m + (j - i) = n$ processes.
   Let P_i be a set of i "real" processes, that is, they are among the m processes of system $\mathcal{S}_m$,[4] and let C be the set of $(j - i)$ fictitious processes of $\mathcal{S}$.
   Now consider any schedule S of the simulated system $\mathcal{S}$. In S it is obvious that the set P_i is timely with respect to itself, and P_i is also timely with respect to the set of crashed processes C. So, by Observation 1, P_i is timely with respect to $P_i \cup C$ in S. Thus, in every schedule S of $\mathcal{S}$, there is a set of size i that is timely with respect to a set of size $i + (j - i) = j$. In other words, every schedule of $\mathcal{S}$ is also a schedule of $\mathcal{S}^i_{j,n}$. So $\mathcal{S} \subseteq \mathcal{S}^i_{j,n}$.
   By assumption, (t, k, n)-agreement can be solved in $\mathcal{S}^i_{j,n}$. Since $\mathcal{S} \subseteq \mathcal{S}^i_{j,n}$, by Observation 5, (t, k, n)-agreement can also be solved in $\mathcal{S}$. Since $\mathcal{S}$ has $(j-i)$ fictitious processes that are permanently crashed, this implies that $(t-(j-i), k, m)$-agreement can be solved in the "real" system $\mathcal{S}_m$. Since $\ell = t-(j-i)$, (ℓ, k, m)-agreement can be solved in $\mathcal{S}_m$. Since $k \leq t - (j - i) = \ell$, (ℓ, ℓ, m)-agreement can be solved in $\mathcal{S}_m$. Since $\mathcal{S}_m$ is the asynchronous system with $m \geq 2$ processes and $1 \leq \ell < m$, this contradicts an impossibility result in [5,7]. Thus, (t, k, n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$. □

As special cases of the above theorem we immediately get that, for $k \leq t$, system $\mathcal{S}^k_{t+1,n}$ is strong enough for solving (t, k, n)-agreement but too weak for solving $(t + 1, k, n)$-agreement and $(t, k - 1, n)$-agreement.

6 Related work

Dwork, Lynch and Stockmeyer [12] introduce the concept of partial synchrony. They propose message-passing models in which there are eventual or unknown bounds on message transmission times and on relative process speeds. These bounds must hold between every pair of processes. It is shown that consensus can be solved in these models. Subsequent work [1,2,13,16–18,20] proposes weaker types of partial synchrony (for message-passing systems) with which consensus can still be solved or Ω can be implemented (Ω is the weakest failure detector for consensus [8]). None of these works consider models in which sub-consensus problems such as (t, k, n)-agreement can be solved, but consensus cannot.

The work in [4] considers a shared-memory model and defines what it means for a single process p to be timely with respect to another process q in any given schedule. The concept of set timeliness introduced in this paper is a generalization of this definition, where individual processes p and q are replaced by sets of processes P and Q.

The IIS model [6] is a round-based model in which, in each round, a process atomically writes a value and obtains a snapshot of the values written by other processes in the round. In this model, set agreement and consensus are impossible. Rajsbaum et al. [23,24] propose a family of models called IRIS that are weaker than the IIS model. This family is parameterized by a property PR_C on the snapshot values that a process can obtain in a round. This property "restricts the asynchrony" of the system, because the fact

[4] Note that there are at least i processes in $\mathcal{S}_m$, because $m = i + (n - j)$ and $n - j \geq 0$.

that a snapshot cannot return certain values means that the execution cannot proceed in certain ways. Specific IRIS models are given in which wait-free k-set agreement is solvable but wait-free $(k-1)$-set agreement is not, thus providing a separation between these problems.

Our model of partial synchrony differs from the IRIS models in two ways. First, we express synchrony behavior directly via timeliness properties of processes, whereas the IRIS models restrict the allowable executions via properties that snapshots must satisfy. Second, our model is based on read-write shared memory, whereas the IRIS model is based on rounds with immediate snapshots. It is possible to implement these rounds in the read-write shared memory model, but it is unclear how the restricted runs of IRIS map to the timeliness properties of the shared memory model. For instance, a process that never appears in the snapshot of other processes may be a process that is actually timely in the shared memory model that implements IRIS: this process may execute at the same speed as other processes but always start a round a few steps later.

Recent work has related some failure detectors with corresponding partially synchronous systems (which can also be viewed as schedulers) [11,22–24]. It turns out that the family of partially synchronous systems $\mathcal{S}^i_{j,n}$ introduced here is also closely related to a natural family of failure detectors. In fact, one can define a failure detector Ω^i_j (a "limited scope" [19] variant of Ω^i [21]) and show that Ω^i_j corresponds to $\mathcal{S}^i_{j,n}$ in the following sense: (1) in a system with Ω^i_j, one can emulate schedules of system $\mathcal{S}^i_{j,n}$, and (2) in system $\mathcal{S}^i_{j,n}$ one can implement Ω^i_j.[5] Failure detector Ω^i_j is defined as follows. At any time, each process trusts some set of size i of the processes; furthermore, there are sets P and Q of processes of size i and j, respectively, and a time after which (1) every correct process in Q trusts P, and (2) every correct process trusts a set containing a correct process.

The problem of k-set agreement was first defined in [10]. The wait-free and t-resilient versions of this problem were shown to have no solutions in asynchronous systems in [5,7,15,25].

Finally, the concept of set timeliness and some of the results presented here first appeared in a preliminary version of this paper in [3] (in that preliminary version, we considered only the partially synchronous systems $\mathcal{S}^i_{n,n}$ and the wait-free version of k-set agreement).

Acknowledgments We are grateful to the anonymous referees for their valuable comments. This work was partially supported by grant ANR-VERSO-SHAMAN and by the National Science and Engineering Research Council of Canada.

[5] We thank an anonymous reviewer for pointing out this correspondence and suggesting a possible definition for Ω^i_j; the Ω^i_j that we define here is simpler but equivalent.

References

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Communication-efficient leader election and consensus with limited link synchrony. In: ACM Symposium on Principles of Distributed Computing, pp. 328–337, July 2004
2. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing Omega in systems with weak reliability and synchrony assumptions. Distrib. Comput. **21**(4), 285–314 (2008)
3. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Partial synchrony based on set timeliness. In: ACM Symposium on Principles of Distributed Computing, pp. 102–110, August 2009
4. Aguilera, M.K., Toueg, S.: Adaptive progress: a gracefully-degrading liveness property. Distrib. Comput. **22**(5–6), 303–334 (2010)
5. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: ACM Symposium on Theory of Computing, pp. 91–100, May 1993
6. Borowsky, E., Gafni, E.: A simple algorithmically reasoned characterization of wait-free computation (extended abstract). In: ACM Symposium on Principles of Distributed Computing, pp. 189–198, August 1997
7. Borowsky, E., Gafni, E., Lynch, N.A., Rajsbaum, S.: The BG distributed simulation algorithm. Distrib. Comput. **14**(3), 127–146 (2001)
8. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. J. ACM **43**(4), 685–722 (1996)
9. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
10. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
11. Cornejo, A., Rajsbaum, S., Raynal, M., Travers, C.: Brief announcement: failure detectors are schedulers. In: ACM Symposium on Principles of Distributed Computing, pp. 308–309, August 2007
12. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
13. Fernández, A., Raynal, M.: From an asynchronous intermittent rotating star to an eventual leader. IEEE Trans. Parallel Distrib. Syst. **21**(9), 1290–1303 (2010)
14. Gafni, E., Kuznetsov, P.: On set consensus numbers. Distrib. Comput. **24**(3–4), 149–163 (2011)
15. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
16. Hutle, M., Malkhi, D., Schmid, U., Zhou, L.: Chasing the weakest system model for implementing Ω and consensus. IEEE Trans. Dependable Secur. Comput. **6**(4), 269–281 (2009)
17. Jiménez, E., Arévalo, S., Fernández, A.: Implementing unreliable failure detectors with unknown membership. Inf. Process. Lett. **100**(2), 60–63 (2006)
18. Malkhi, D., Oprea, F., Zhou, L.: Omega meets Paxos: leader election and stability without eventual timely links. In: International Conference on Distributed Computing. LNCS, vol. 3724, pp. 199–213. Springer, September 2005
19. Mostéfaoui, A., Raynal, M.: k-set agreement with limited accuracy failure detectors. In: ACM Symposium on Principles of Distributed Computing, pp. 143–152, July 2000
20. Mostefaoui, A., Raynal, M., Travers, C.: Time-free and timer-based assumptions can be combined to obtain eventual leadership. IEEE Trans. Parallel Distrib. Syst. **17**(7), 656–666 (2006)
21. Neiger, G.: Failure detectors and the wait-free hierarchy. In: ACM Symposium on Principles of Distributed Computing, pp. 100–109, August 1995
22. Pike, S.M., Sastry, S., Welch, J.L.: Failure detectors encapsulate fairness. In: International Conference on Principles of Distributed Systems. LNCS, vol. 6490, pp. 173–188. Springer, December 2010

23. Rajsbaum, S., Raynal, M., Travers, C.: Failure detectors as schedulers (an algorithmically-reasoned characterization). Technical Report 1838, IRISA, Université de Rennes, France, March 2007
24. Rajsbaum S., Raynal M., Travers C.: The iterated restricted immediate snapshot model. In: International Computing and Combinatorics Conference. LNCS, vol. 5092, pp. 487–497. Springer, June 2008
25. Saks, M.E., Zaharoglou, F.: Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)
26. Zielinski, P.: Anti-Ω: the weakest failure detector for set agreement. Distrib. Comput. **22**(5–6), 335–348 (2010)