Chasing the Weakest System Model for Implementing Ω and Consensus

Martin Hutle, Dahlia Malkhi, Ulrich Schmid, Lidong Zhou

Abstract—Aguilera et al. and Malkhi et al. presented two system models, which are weaker than all previously proposed models where the eventual leader election oracle Ω can be implemented and thus also consensus can be solved. The former model assumes unicast steps and at least one correct process with f outgoing eventually timely links, whereas the latter assumes broadcast steps and at least one correct process with f bidirectional but moving eventually timely links. Consequently, those models are incomparable. In this paper, we show that Ω can also be implemented in a system with at least one process with f outgoing moving eventually timely links, assuming either unicast or broadcast steps. It seems to be the weakest system model that allows to solve consensus via Ω-based algorithms known so far. We also provide matching lower bounds for the communication complexity of Ω in this model, which are based on an interesting "stabilization property" of infinite runs. Those results reveal a fairly high price to be paid for this further relaxation of synchrony properties.

Index Terms—Distributed Systems, Failure Detectors, Fault-tolerant Distributed Consensus, System Modeling, Partial Synchrony.

1 Introduction

The chase for the weakest system model that allows to solve consensus has long been an active branch of research in distributed algorithms. To circumvent the FLP impossibility in asynchronous systems [16], many models in between synchrony and asynchrony [13] have been proposed over the years: The Archimedean model [28], the classic partially synchronous models [10], [14], the semi-synchronous models [7], [25], the Θ-Model [19], [29], the model of [24] and the FAR-Model [15].

Another recent branch of this research is the chase [1], [4], [5], [6], [20], [21] for the weakest system model that allows the implementation of the eventual leader oracle failure detector Ω. For a system of n partially synchronous [14] or semi-synchronous [25] processes where at most f may crash, a suite of models with more and more relaxed link synchrony assumptions has been developed. In this setting, a link between two processes is called timely at time t if a message sent at time t is not received later than $t+\delta$. The bound δ can be known or unknown. A link is called timely (resp. eventually timely) if it is timely at all times $t \geq 0$ (resp. $t \geq t_{GST}$, for some unknown time t_{GST}). A link is *fair lossy*, if, for each class of messages, infinitely many messages are sent over this link, infinitely many of them are received.

The following models have been proposed so far:

Stable Leader Election [4], denoted $\mathcal{S}^{\leftrightarrow}_{n-1}$: At least one correct process must have $n-1$ eventually timely bidirectional links with known delay bound δ (the other links can be purely asynchronous). All links except the timely ones are fair lossy. A message to at most one process can be sent in a single computing step (unicast steps).

Aguilera et al. 2003 [1], denoted $\mathcal{S}^{\rightarrow}_{n-1}$: At least one correct process must have $n-1$ eventually timely outgoing links ($\diamond(n-1)$-source) with unknown delay bound δ. All links except the timely ones are fair lossy, and computing steps are unicast.

Aguilera et al. 2004 [5], denoted $\mathcal{S}^{\rightarrow}_{f}$: At least one correct process must have f eventually timely outgoing links ($\diamond f$-source) with unknown delay bound δ. All links except the timely ones are fair lossy, and computing steps are unicast.

Malkhi et al. [20], denoted $\mathcal{S}^{\leftrightarrow}_{f*}$: There must be at least one correct process with links that eventually permit a bounded-delay round-trip with at least f neighbors (f-accessibility), at any time. The set of neighbors may change over time (i.e., may be moving [26], [27]). All links must be reliable and the delay bound δ must be known (but the result can be extended to an unknown bound and message loss). A message may be sent to all processes in a single computing step (broadcast steps).

Note that Ω with stability properties can be built in $\mathcal{S}^{\leftrightarrow}_{n-1}$, and in $\mathcal{S}^{\leftrightarrow}_{f*}$ for certain values of f.

An orthogonal approach looks at the weakest failure detector abstraction to uphold Ω and consensus. Limited scope FDs [6], [11], [17], [21], [22], [23], [30] require only f processes other than the leader to uphold the accuracy requisite. Although not concerned with implementation, it is clear that an x-accurate FD requires that at least one correct process must have x eventually timely outgoing links with unknown delay bound. Hence, these abstractions require $\mathcal{S}^{\rightarrow}_{f}$ as well.

This is a slightly extended version of the original paper that appeared in IEEE TDSC 6(4):269-281, 2009, which adds some omitted technical details in the proofs of Lemma 8 and 9.

- *Martin Hutle is with the LSR group, EPFL, CH-1015 Lausanne, Switzerland. Email: martin.hutle@epfl.ch.*
- *Dahlia Malkhi and Lidong Zhou are with Microsoft Research. Email: dahlia@microsoft.com, lidongz@microsoft.com.*
- *Ulrich Schmid is with the Embedded Computing Systems Group, Technische Universität Wien, Austria. Email: s@ecs.tuwien.ac.at.*

We can summarize the differences between the existing models as follows:

Name	$\diamond$timely	pattern	link	steps
$\mathcal{S}^{\leftrightarrow}_{n-1}$	$n-1$	—	bidir.	unicast
$\mathcal{S}^{\rightarrow}_{n-1}$	$n-1$	—	outgoing	unicast
$\mathcal{S}^{\rightarrow}_{f}$	f	fixed	outgoing	unicast
$\mathcal{S}^{\leftrightarrow}_{f*}$	f	moving	bidir.	bcast

In the sequel, let $\mathcal{A} \subseteq \mathcal{B}$ denote the fact that model $\mathcal{B}$ is weaker than model $\mathcal{A}$ w.r.t. inclusion of sets of executions, such that every correct Ω implementation for model $\mathcal{B}$ also works correctly in $\mathcal{A}$. We obviously have: $\mathcal{S}^{\leftrightarrow}_{n-1} \subseteq \mathcal{S}^{\rightarrow}_{n-1} \subseteq \mathcal{S}^{\rightarrow}_{f}$. W.r.t. link synchrony, $\mathcal{S}^{\leftrightarrow}_{f*}$ is also weaker than any of $\mathcal{S}^{\leftrightarrow}_{n-1}$, $\mathcal{S}^{\rightarrow}_{n-1}$and $\mathcal{S}^{\rightarrow}_{f}$. However, $\mathcal{S}^{\leftrightarrow}_{f*}$ requires bidirectional links and uses more powerful computing steps, hence is stronger than any of $\mathcal{S}^{\leftrightarrow}_{n-1}$, $\mathcal{S}^{\rightarrow}_{n-1}$and $\mathcal{S}^{\rightarrow}_{f}$. As a consequence, $\mathcal{S}^{\leftrightarrow}_{f*}$ and the latter models are incomparable.

From the above relations, it follows that $\mathcal{S}^{\leftrightarrow}_{f*}$ and $\mathcal{S}^{\rightarrow}_{f}$ are currently the weakest system models for implementing Ω and hence consensus. One may wonder, however, whether a system that is like $\mathcal{S}^{\rightarrow}_{f}$, except that the timely links of the $\diamond f$-source may be moving like in $\mathcal{S}^{\leftrightarrow}_{f*}$, is strong enough for this problem? Stated in the terminology of $\mathcal{S}^{\leftrightarrow}_{f*}$: Whether it is sufficient for the f links that ensure $\diamond f$-accessibility to be timely only in the outgoing direction? This paper answers the above question in the affirmative.

Detailed contributions:

1) We define a system model $\mathcal{S}^{\rightarrow}_{f*}$, which can be simulated in two models $\mathcal{S}^{b\rightarrow}_{f*}$ and $\mathcal{S}^{u\rightarrow}_{f*}$ that are weaker than $\mathcal{S}^{\leftrightarrow}_{f*}$ and $\mathcal{S}^{\rightarrow}_{f}$, respectively. All those models require at least one correct process that is a $\diamond$moving-f-source: It must have timely outgoing links with an unknown delay bound to a moving set of f receivers at any time. We also provide a simple Ω implementation for $\mathcal{S}^{\rightarrow}_{f*}$, which hence allows to implement Ω in $\mathcal{S}^{b\rightarrow}_{f*}$ and $\mathcal{S}^{u\rightarrow}_{f*}$ as well.
2) We prove a lower bound, which shows that $\Omega(nf)$ links must carry messages forever in any correct Ω implementation for $\mathcal{S}^{\rightarrow}_{f*}$.
3) We also give a communication-optimal Ω implementation for $\mathcal{S}^{\rightarrow}_{f*}$ (for reliable links), where $O(nf)$ links carry messages forever.
4) We show that the above communication-optimal Ω implementation for $\mathcal{S}^{\rightarrow}_{f*}$ lets only $n-1$ links carry messages forever in the case where the $\diamond$moving-f-source eventually stabilizes to a non-moving $\diamond(n-1)$-source, as in [1].
5) We show that $n-1$ links must carry messages forever in any such Ω implementation.

2 System Model $\mathcal{S}^{\rightarrow}_{f*}$

Since the models $\mathcal{S}^{\rightarrow}_{f}$ and $\mathcal{S}^{\leftrightarrow}_{f*}$ are "structurally different", in the sense that they use different basic computing steps, there is no single model that is weaker than both $\mathcal{S}^{\rightarrow}_{f}$ and $\mathcal{S}^{\leftrightarrow}_{f*}$ at the same time, in the sense of inclusion of sets of executions. However, every algorithm that works in a unicast model also works in a broadcast model (with minor changes). Moreover, in the unicast model, it is usually possible to simulate broadcast steps via multiple unicast steps. Hence, we will start out from a model with broadcast steps and derive a model with unicast steps from it.

More specifically, the algorithms, correctness proofs and lower bound results developed in this paper will be based on a system model $\mathcal{S}^{\rightarrow}_{f*}$, which assumes broadcast steps, at least one $\diamond$moving-f-source, and reliable links (see Section 2.3 for details). In order to prove our claim that Ω can be implemented in models that are weaker than $\mathcal{S}^{\rightarrow}_{f}$ and $\mathcal{S}^{\leftrightarrow}_{f*}$, we will show that algorithms designed for $\mathcal{S}^{\rightarrow}_{f*}$ can be transformed to work also in two additional models $\mathcal{S}^{u\rightarrow}_{f*} \supseteq \mathcal{S}^{\rightarrow}_{f}$ and $\mathcal{S}^{b\rightarrow}_{f*} \supseteq \mathcal{S}^{\leftrightarrow}_{f*}$. The model $\mathcal{S}^{u\rightarrow}_{f*}$ (see Section 2.2) assumes unicast steps and is weaker than $\mathcal{S}^{\rightarrow}_{f}$, whereas $\mathcal{S}^{b\rightarrow}_{f*}$ (see Section 2.1) assumes broadcast steps and is weaker than $\mathcal{S}^{\leftrightarrow}_{f*}$.

Using the notation that model B is weaker than A w.r.t. simulation ($A \subseteq_s B$) when there is a simulation that allows an algorithm for B to be run in A, we will hence show that $\mathcal{S}^{u\rightarrow}_{f*} \subseteq_s \mathcal{S}^{\rightarrow}_{f*}$ and $\mathcal{S}^{b\rightarrow}_{f*} \subseteq_s \mathcal{S}^{\rightarrow}_{f*}$. Although of course $\mathcal{S}^{u\rightarrow}_{f*} \not\subseteq \mathcal{S}^{\rightarrow}_{f*}$ and $\mathcal{S}^{b\rightarrow}_{f*} \not\subseteq \mathcal{S}^{\rightarrow}_{f*}$ w.r.t. set inclusion, it is nevertheless true that any of our algorithms for $\mathcal{S}^{\rightarrow}_{f*}$ in conjunction with the appropriate simulation leads to a correct implementation of Ω in $\mathcal{S}^{u\rightarrow}_{f*}$ resp. $\mathcal{S}^{b\rightarrow}_{f*}$. Moreover, all lower bound results for $\mathcal{S}^{\rightarrow}_{f*}$ developed in this paper will carry over to $\mathcal{S}^{u\rightarrow}_{f*}$ and $\mathcal{S}^{b\rightarrow}_{f*}$ as well.

All our models assume a fully-connected network of n processes Π, f of which may fail by crashing. Let $\mathcal{C} \subseteq \Pi$ be the set of correct processes, i.e. processes that never fail. Every process executes an algorithm consisting of atomic computing steps. Like in [5], [14], we assume that processes are partially synchronous, in the sense that every non-crashed process takes at least one step every Φ steps of the fastest process. The speed ratio bound Φ is unknown to the algorithm. To make our models as weak as possible, we do not assume real-time clocks as in [5][1], but rather adopt the convention from [14] that real-time is measured in multiples of the steps of the fastest process. In particular, the (unknown) delay bound δ is such that the fastest—and hence any—process can take at most δ steps while a timely message is in transit. Hence, we can use simple step-counting for timing out messages.

2.1 Broadcast model $\mathcal{S}^{b\rightarrow}_{f*}$

In every step, a process may broadcast at most one message to every other process in the system. If not specified otherwise, links are fair lossy [5].

1. It it is easy to build our algorithms and proofs on real-time clocks as well. However, since the source has to be at least partial synchronous anyway, this would be a stronger assumption.

Definition 1: In the broadcast model, we say that a unidirectional link (p, q) is *timely at time* t if no message broadcast by p at time t is received at q after time $t + \delta$.

Note that a timely link/message does not require the receiver to be correct; if the receiver is faulty, then it is vacuously considered timely. Also, when no message is sent at time t, the link/message is considered timely at time t.

Definition 2: In the broadcast model, a process p is a *moving-j-source at time t* if there exist j other processes q such that the links (p, q) are timely at t.

Definition 3: A process p is a *$\diamond$moving-j-source* if it is correct and there is a time t such that, for all $t' \geq t$, p is a moving-j-source. It is called a *perpetual moving-j-source* if this holds for $t' \geq 0$.

In $\mathcal{S}^{b\rightarrow}_{f*}$, we require that there is at least one $\diamond$moving-f-source. Note that if δ is unknown and the links are reliable, then a $\diamond$moving-j-source is also a moving-j-source (with some different δ) for all $t \geq 0$.

Since a $\diamond j$-accessible process [20] has j moving bidirectional timely links, it follows immediately:

Corollary 1: A $\diamond j$-accessible process is also a $\diamond$moving-j-source.

2.2 Unicast model $\mathcal{S}^{u\rightarrow}_{f*}$

In every step, a process may send at most one message to some other process in the system. If not specified otherwise, links are fair lossy [5].

We assume that every message sent by a process in $\mathcal{S}^{u\rightarrow}_{f*}$ belongs to exactly one message class. A message class $\mathcal{M}_p$ is said to be of *broadcast type* if, in every execution σ, p sends all messages $\in \mathcal{M}_p$ to its $n-1$ peer processes in strict round-robin order. For example, the sequence of receivers of all send steps of p in σ where a message in $\mathcal{M}_p$ is sent may be $p_2, p_3, \ldots, p_n, p_2, p_3, \ldots, p_n, p_2, p_3 \ldots$, where $\{p_2, \ldots, p_n\} = \Pi - \{p\}$. The k-th occurrence of a group of of $n-1$ consecutive sends (to all its peer processes) in such a sequence is called p's k-th u-broadcast of $\mathcal{M}_p$, and it is said to occur at the time t when the first of its send steps occurs.

Definition 4: In $\mathcal{S}^{u\rightarrow}_{f*}$, a process p is a moving-j-source at time t if no more than $n-j-1$ of the $n-1$ messages sent in a single u-broadcast at time t are received after time $t+\delta$. We then say that the at least j other messages are *timely at time t*. No assumption is made about the timeliness of messages that are not u-broadcast.

In $\mathcal{S}^{u\rightarrow}_{f*}$, we require that there is at least one $\diamond$moving-f-source according to Definition 3 (with "moving-j-source" defined according to Definition 4). The model is hence such that it has purely asynchronous links, except for messages which are u-broadcast by the $\diamond$moving-f-source after stabilization time: Any such message finds at least f timely links, which may be different for different messages. Of course, nothing prevents an algorithm from not u-broadcasting any message in $\mathcal{S}^{u\rightarrow}_{f*}$, but then there are only purely asynchronous links.

Since a $\diamond j$-source [5] has f *fixed* links that are eventually always timely, it follows immediately:

Corollary 2: A $\diamond j$-source is also a $\diamond$moving-j-source.

Putting everything together, we have:

Corollary 3: $\mathcal{S}^{\rightarrow}_{f} \subseteq \mathcal{S}^{u\rightarrow}_{f*}$ and $\mathcal{S}^{\leftrightarrow}_{f*} \subseteq \mathcal{S}^{b\rightarrow}_{f*}$.

2.3 The broadcast model $\mathcal{S}^{\rightarrow}_{f*}$

The model $\mathcal{S}^{\rightarrow}_{f*}$, which will be used for our algorithms and lower bounds, is identical to the broadcast model $\mathcal{S}^{b\rightarrow}_{f*}$, except that it assumes reliable links. Hence, Definitions 1–3 apply also to $\mathcal{S}^{\rightarrow}_{f*}$. To disambiguate between $\mathcal{S}^{\rightarrow}_{f*}$ and $\mathcal{S}^{b\rightarrow}_{f*}$, we will use the term *send-to-all* to denote a broadcast in $\mathcal{S}^{\rightarrow}_{f*}$.

We will now sketch how $\mathcal{S}^{\rightarrow}_{f*}$ can be simulated in $\mathcal{S}^{b\rightarrow}_{f*}$ and in $\mathcal{S}^{u\rightarrow}_{f*}$. First, send-to-all is mapped to a true broadcast step in the broadcast model, and to $n-1$ consecutive sends to all peer processes in the unicast model. Moreover, send-to-all involves a protocol for reliable communication atop of fair lossy links. There are advanced techniques like [8] or [2] that could be adopted here, but even a simple piggyback and acknowledgment protocol suffices: Any message from p that has not been acknowledged by receiver q is piggybacked on the next messages sent to q in $\mathcal{S}^{u\rightarrow}_{f*}$ (resp. broadcast in $\mathcal{S}^{b\rightarrow}_{f*}$), so that it is retransmitted until an acknowledgment message has been received. Acknowledgment messages are also piggybacked on the messages sent (resp. broadcast) by q, and hence retransmitted until p does no longer piggyback the message. It is obvious that this protocol implements reliable communication between any pair of correct processes. Note that a message that was lost will not become timely by this simulation. But, for messages that are timely, the simulation preserves the timeliness of the message:

Lemma 1: With our simulations, a $\diamond$moving-f-source in $\mathcal{S}^{u\rightarrow}_{f*}$ resp. in $\mathcal{S}^{b\rightarrow}_{f*}$ is also a $\diamond$moving-f-source in $\mathcal{S}^{\rightarrow}_{f*}$. Its delay bound is $\delta' = \delta$ for the simulation in $\mathcal{S}^{b\rightarrow}_{f*}$, and $\delta' = \delta + (n-2)\Phi$ for the simulation in $\mathcal{S}^{u\rightarrow}_{f*}$.

Proof: We can restrict our attention to timely messages, since non-timely messages are not adversely affected by retransmissions. Timely messages are never lost by definition, however, so no retransmission is actually necessary here.

For the simulation in the broadcast model $\mathcal{S}^{b\rightarrow}_{f*}$, a timely message in $\mathcal{S}^{b\rightarrow}_{f*}$ obviously remains timely (with $\delta' = \delta$) in $\mathcal{S}^{\rightarrow}_{f*}$. For the simulation in the unicast model $\mathcal{S}^{u\rightarrow}_{f*}$, the implementation of send-to-all as $n-1$ unicast sends of the same message to all peer processes causes every message class to be of broadcast type in $\mathcal{S}^{u\rightarrow}_{f*}$. Consequently, by Definition 4, at most $n-f-1$ of the $n-1$ unicast messages sent in any send-to-all may be received later than δ time after the first unicast send. Due to our partially synchronous processes, performing $n-1$ consecutive send steps takes at most $(n-2)\Phi$ time in $\mathcal{S}^{u\rightarrow}_{f*}$, thus $\delta' = \delta + (n-2)\Phi$ is a bound on the delay of timely messages in $\mathcal{S}^{\rightarrow}_{f*}$. □

Remark. One could argue that, in practice, the retransmission of messages could compromise their timeliness, due to the potentially linearly-growing message size caused by the piggybacking mechanism.

Fortunately, with our algorithms, this can be avoided due to the fact that only some of the messages, namely, the ALIVE messages (cf. e.g. Algorithm 1, Section 3), need to be timely, but not reliable (in case they are not timely), whereas all the other messages never need to be timely, but always reliable. Thus, the piggybacking mechanism needs to be employed only for messages that are not ALIVE messages. On the other hand, ALIVE messages can just be sent as unreliable datagrams, preferably out-of-band, i.e., with priority over ordinary messages, such that even a large number of messages in the buffers does not affect the transmission time of ALIVE messages.

We decided not to incorporate this differentiation of messages in our system model, since we are primarily concerned about solvability issues in this paper. To improve readability, we hence just assumed that $\mathcal{S}^{\rightarrow}_{f*}$ provides reliable links. It should be straightforward to extend our results when needed, however.

3 Ω IMPLEMENTATION IN $\mathcal{S}^{\rightarrow}_{f*}$

We consider the definition of the Ω failure detector as introduced in [9]:

Definition 5: A failure detector is of class Ω if the failure detector output $H(p,t) \in \Pi$ and the following holds:

Eventual Leadership: There is a time after which all the correct processes always trust the same correct process. Formally,

$$\exists t, \exists \ell \in \mathcal{C}, \forall p \in \mathcal{C}, \forall t' \geq t : \; H(p,t') = \ell$$

In this section, we introduce and analyze a simple implementation of Ω in $\mathcal{S}^{\rightarrow}_{f*}$, given as Algorithm 1. It is well-suited for explaining the principle of operation, but suffers from ever increasing timeouts. In Section 5, we also provide an algorithm where the timeout values stabilize for all links that are eventually timely.

The algorithm works as follows: Every process p periodically sends-to-all ALIVE messages every η steps (line 8). These messages are equipped with a monotonically increasing sequence number seq_p. The receiver task (Task 2) of process q waits for and collects ALIVE messages with a sequence number equal to the current receiver sequence number $rseq_q$. A timer with a timeout of Δ_q steps is used for terminating the wait; both $rseq_q$ and Δ_q are incremented when the timer expires (line 20). Note that this timer can be implemented via step counting, since we assume partially synchronous processes. Although processes are not synchronized, i.e., there is no a priori correlation between the time any process sends an ALIVE message with a certain sequence number and any receiver's timeout for that sequence number, using an ever increasing Δ_q will allow q to eventually get every timely ALIVE message from p before the timeout (we will call this *timely received* in the sequel).

Algorithm 1 Implementation of Ω in $\mathcal{S}^{\rightarrow}_{f*}$, code for process p

Variables
1: $\forall q \in \Pi :\ counter_p[q] \in \mathbb{N}$, initially 0
2: $\forall q \in \Pi\ \forall s \in \mathbb{N} :\ suspect_p[q][s] \subseteq \Pi$, initially $\emptyset$
3: $\forall q \in \Pi\ \forall s \in \mathbb{N} :\ reported_p[q][s] \in \{\texttt{true}, \texttt{false}\}$, initially `false`
4: $seq_p, rseq_p \in \mathbb{N}$, initially 0
5: Δ_p, initially η

Initially
6: start Tasks 1,2
7: start $timer$ with Δ_p

Task 1
8: **every** η steps **do**
9: send $\langle \text{ALIVE}, p, seq_p, counter_p \rangle$ to all $q \in \Pi$
10: $seq_p \leftarrow seq_p + 1$
11: $leader_p \leftarrow q$, where $\langle counter_p[q], q \rangle$ is minimal

Task 2
12: **upon** receive $\langle \text{ALIVE}, q, s, c \rangle$ **do**
13: $reported_p[q][s] \leftarrow \texttt{true}$
14: **for all** $r \in \Pi$ **do**
15: $counter_p[r] \leftarrow \max\{counter_p[r], c[r]\}$

16: **upon** expiration of $timer$ **do**
17: **for all** q, where $reported_p[q][rseq_p] = \texttt{false}$ **do**
18: send $\langle \text{SUSPECT}, q, p, rseq_p \rangle$ to all $r \in \Pi$ (including p itself)
19: $reported_p[q][rseq_p] \leftarrow \texttt{true}$
20: $rseq_p \leftarrow rseq_p + 1$
21: $\Delta_p \leftarrow \Delta_p + 1$
22: set $timer$ to Δ_p

23: **upon** receive $\langle \text{SUSPECT}, q, r, s \rangle$ **do**
24: $suspect_p[q][s] \leftarrow suspect_p[q][s] \cup \{r\}$
25: **if** $|suspect_p[q][s]| \geq n - f$ **then**
26: $counter_p[q] \leftarrow counter_p[q] + 1$

Every receiver process q maintains an array $counter_q[p]$, which essentially contains the number of suspicions of sender p encountered at q so far: The sender p is suspected at q if q is notified of the fact that at least $n - f$ receivers experienced a timeout for the same sequence number s. This notification is done via SUSPECT messages, which are sent-to-all (line 18) by any receiver process that experienced a timeout for sender p with sequence number s. For the latter, q maintains a binary array $reported_q[p][rseq]$, which records whether an ALIVE message from p has been received by the timeout for sequence number $rseq_q$ or not.

The receiver q increases $counter_q[p]$ if it receives at least $n-f$ of such SUSPECT messages for p with sequence number s (line 25). In addition, $counter_q[p]$ is increased if a larger counter value for process p is observed in any ALIVE message (line 15). Note that $counter_q[p]$ may be raised by more than 1 here. The process $p = \ell$ with minimal counter value in $counter_q[p]$ (or the minimal process id in case of several such entries) is elected as q's leader (line 11). The array $suspect_q[p][s]$ is used to store the set of processes from which suspect messages for sender p and sequence number s have been received so far (line 24).

Informally, the correctness of Algorithm 1 follows from the following reasoning: At the time the $\diamond$moving-f-source becomes a moving-f-source, at least f outgoing links of the source p carry timely messages at any time. Thus, eventually, it is impossible that the quorum of $n-f$ SUSPECT messages is reached for p for any sequence number. Note that this even holds true if some of the f timely receiver processes have crashed. Consequently, all processes stop increasing the counter for process p, whereas the counter of every crashed sender process keeps increasing forever since every receiver obviously experiences a timeout here. Since the counter values are continuously exchanged via the content of the ALIVE messages, eventually all processes reach agreement on all counters that have stopped increasing. Hence, locally electing the process with minimal counter indeed leads to a correct implementation of Ω.

The detailed proof of correctness below follows the proof in [5]:

Lemma 2: For every processes p and q, $counter_p[q]$ as well as the timeout Δ_q are monotonically nondecreasing.

Proof: Clear from the way $counter_p[q]$ and Δ_q are updated. □

Lemma 3: If p is correct and q is faulty, then $counter_p[q]$ is unbounded.

Proof: If q is faulty, then eventually it stops sending $\langle \text{ALIVE}, p, seq_p, c\rangle$ messages. Therefore there is a sequence number s, such that for all $s' \geq s$, every correct process p', $reported_{p'}[q][s']$ is never set to `true` in line 13, and thus p' sends a $\langle \text{SUSPECT}, q, p', s'\rangle$ to all processes (line 18). Since there are at least $n-f$ correct processes, it follows that p increments $counter_p[q]$, and since there are infinitely many s', this happens infinitely often. □

The following Lemma 4 shows that, irrespectively of how far sender and receiver have been out of sync initially, a correct receiver eventually gets all timely messages with a certain sequence number before it timeouts this sequence number.

Lemma 4: Let σ_p^k denote the time where p sends $\langle \text{ALIVE}, p, k, *\rangle$ and ρ_q^k be the time where q timeouts message k, i.e., increments the receiver sequence number $rseq_q$ from k to $k+1$. Then $\sigma_p^k - \rho_q^k$ is strictly monotonically decreasing, without bound, from some $k \geq k_0$ on.

Proof: For all $k > 0$, $\sigma_p^k \leq \sigma_p^{k-1} + \Phi\eta$, whereas $\rho_q^k \geq \rho_q^{k-1} + \eta + k$. Now, for any k, $\sigma_p^k - \rho_q^k \leq (\sigma_p^{k-1} + \Phi\eta) - (\rho_q^{k-1} + \eta + k) = \sigma_p^{k-1} - \rho_q^{k-1} + (\Phi - 1)\eta - k$. Expanding this yields $\sigma_p^k - \rho_q^k \leq \sigma_p^0 - \rho_q^0 + k(\Phi-1)\eta - k(k+1)/2$, which proves the lemma. □

Lemma 5: If p is a correct $\diamond$moving-f-source then, for every process q, $counter_q[p]$ is bounded.

Proof: By way of contradiction, assume that $counter_q[p]$ is unbounded. Then there is at least one process r, where line 25 is true infinitely often for p, which requires $n-f$ $\langle \text{SUSPECT}, p, *, s\rangle$ for infinitely many s. Since p is a $\diamond$moving-f-source, there is a sequence number s', so that for every $s \geq s'$, there is a set $Q(t_s)$ of at least f processes that receive $\langle \text{ALIVE}, p, s, c\rangle$ by time δ after it was sent at time t_s. Since p never suspects itself, there must be at least one process $r_s \in Q(t_s)$ that sends $\langle \text{SUSPECT}, p, *, s\rangle$ in order to reach the $n-f$ quorum, despite of the fact that $\langle \text{ALIVE}, p, s, *\rangle$ was received timely at r_s, for infinitely many s. Since there are only finitely many processes in the system, there must be at least one correct process r that occurs infinitely often among the processes r_s. This process must hence infinitely often time out a timely message. This is impossible because of Lemma 4, however, which shows that there is some k ensuring $\forall k' > k : \sigma_p^{k'} + \delta < \rho_q^{k'}$. □

Lemma 6: Let $counter_p^t[q]$ denote the value of $counter_p[q]$ at process p at time t. If p and q are correct processes then, for every time t and every correct process r, there exists a time after which $counter_r[q] \geq counter_p^t[q]$.

Proof: Let $x = counter_p^t[q]$. If $x = 0$ or $p = r$ then the result follows from Lemma 2, and so assume $x > 0$ and $p \neq r$. After p sets $counter_p[q] = x$, it sends infinitely many ALIVE messages to all other processes. By Lemma 2, the value of $c[q]$ in these messages is at least x. Since links are fair lossy (even reliable), process r eventually receives at least one of these messages, and sets $counter_r[q]$ to a value that is at least x if $counter_r[q]$ has a smaller value. Afterwards, from Lemma 2, $counter_r[q] \geq x$. □

Lemma 7: If p is correct and there is a time after which $counter_p[q] = x$ forever, then, for every correct process r, there is a time after which $counter_r[q] = x$.

Proof: Follows immediately from Lemma 6, since both p and r are correct. □

Theorem 1: Algorithm 1 implements Ω in system $\mathcal{S}_{f*}^{\rightarrow}$

Proof: By Lemmas 2, 5 and 3, and the fact that there exists a correct $\diamond$moving-f-source, it follows that for every correct process p there is a time after which $leader_p$ is correct and stops changing. By Lemma 7, for every correct processes p and q, there is a time after which $leader_p = leader_q$. □

Hence, $\mathcal{S}_{f*}^{\rightarrow}$ allows to implement Ω. On the other hand, from Theorem 16 in [5], we know:

Theorem 2: [5] There is no Ω implementation in $\mathcal{S}_j^{\rightarrow}$, where $j < f$.

Together with Corollary 2, we thus have:

Corollary 4: There is no Ω implementation in $\mathcal{S}_{j*}^{\rightarrow}$, where $j < f$.

4 COMMUNICATION COMPLEXITY

In this section we consider message complexity, in the following sense: Obviously, Algorithm 1 continuously sends ALIVE messages over all links. In terms of the communication complexity measure introduced by Larrea et. al. in [18], namely, counting the worst-case number of communication channels that are used infinitely often throughout some execution run, the algorithm has complexity $\Omega(n^2)$.

4.1 Lower bounds

Aguilera et al. [5] provided a variant of their algorithm, where under the additional assumption of reliable links,

only f links are required to carry messages forever in some execution. In contrast to this, we show that no such implementation exists if we have only a moving-f-source. Our (much higher) lower bound obviously also applies to the case where we have only a $\diamond$moving-f-source, which is weaker than a moving-f-source. Note that we restrict our attention to $n > f+1$ in this section, since a moving-f-source and an f-source are the same in case of $n = f+1$. After all, moving is meaningless if there are only f outgoing links.

In the sequel, we consider runs R of a correct Ω implementation in such a system. Any such run consists of an infinite number of steps, which can either be a computing or a receive step of some process. A receive step just models the arrival of a message at the receiver; we assume that processing of the received message occurs in the next computing step. The delay bound δ for timely messages according to Definition 1 refers to the time interval from the sender p's computing step (at time t_p^s) until the receiver q's receive step (at time t_q^r). Note that if a computing step (typically triggered by a timeout in our algorithms) at process q occurs at some time $t_q \geq t_p^s + \delta$, then one can be sure that the message has (already) been processed by time t_q.

An admissible run R is a run where every process except the at most f faulty ones performs infinitely many steps in R, and where all messages sent to correct processes are eventually received. Finally, a *feasible run* denotes an admissible run R of a correct Ω implementation in a system with certain properties—like the one that at least one process is a $\diamond$moving-f-source, or that every process is a moving-f-source, depending on the context. In a feasible run R, the eventually elected leader will be denoted $\ell(R)$, and the simple directed graph induced by the set of links that eventually carry messages forever will be called the *effective communication graph* $G_e(R)$. We are interested in the worst-case number of links in $G_e(R)$ over all feasible executions R.

It is important to note that a restricted effective communication graph ($G_e(R) \neq$ fully connected graph) reduces the number of timely messages received from a moving-f-source: The links carrying timely messages at some time t and the links in $G_e(R)$ may be different. Since an unused link is also counted as timely in our setting, only the links in the intersection of those sets need to deliver timely messages to their receivers. Note that this is not an artificial assumption, but rather the faithful modeling of the fact that delays are under the control of the system adversary, whereas the effective communication graph is under the control of the algorithm.

As our main lower bound results, we will prove that the effective communication graph $G_e(R)$ must contain (1) $n-1$ outgoing links from the eventual leader, and (2) a total of $\Omega(nf)$ links in some run R. Comparison with [5] hence reveals an interesting trade-off between weaker synchrony assumptions and communication efficiency. The intuition behind these lower bounds is that at least $n-f$ processes must suspect a moving-f-source before it can actually be considered faulty (and hence demoted in case it is the leader). Otherwise, a process's suspicion may just be a consequence of currently not being in the moving set of timely receivers. However, since the set of processes that receives timely messages may change forever, processes have to continuously inform all other processes about their suspicions regarding the current leader. For the eventual leader, this implies the need to send to all its $n-1$ peer processes. Moreover, exchanging suspicions among all processes in an f-resilient way requires $\Omega(nf)$ links to be used forever.

For our formal proof, we need a notation that allows us to relate runs R and R' to each other that eventually differ only in the moving pattern and communication delays. To simplify notation, time will be just the number of steps in a run here: The finite prefix R_t of a run $R = (s_1, s_2, \dots)$ up to time t is hence just $R_t = (s_1, s_2, \dots\ s_t)$. The extension of a prefix R_t leading to run R is the suffix $E_t = R \setminus R_t = (s_{t+1}, s_{t+2}, \dots)$, and a finite extension of R_t is a prefix of some E_t.

Definition 6: In a system with at least one $\diamond$moving-f-source, an extension E_t of some given prefix R_t is called *feasible* if $R = R_t E_t$ is a feasible run, and it is called *stable* iff no further process crashes and status changes of any $\diamond$moving-f-source occur in the feasible extension E_t. Any two stable extensions E_t and E_t' of the same prefix R_t, as well as the runs $R = R_t E_t$ and $R' = R_t E'$, are called *R_t-similar*, or just *similar* if R_t is clear from the context.

Note carefully that, when considering R_t-similar runs R and R', we always assume that both R and R' are stable, i.e., that all crashes have already occured in R_t and that every $\diamond$moving-f-source has already become a perpetual moving-f-source in E_t. Given that those events must of course occur within finite time, assuming stability in this respect is not a restriction.

The following two lemmas are instrumental in our lower bound proofs. Actually, they are of independent interest, since they reveal an interesting additional stabilization property of infinite runs of correct Ω implementations in $\mathcal{S}_{f*}^{\rightarrow}$: For example, Lemma 8 assures that, starting from any finite prefix P, there is a finite extension R_t after which the system has stabilized, in the sense that the leader does not change anymore. Of course, events that happen within finite time, like process crashes and all $\diamond$moving-f-source status changes, have already occurred within R_t due to the stability assumption discussed before. The surprising fact, however, is that the stabilization of the leader happens in presence of events like moving pattern and timing changes, which occur infinitely often, hence also after R_t.

Lemma 8: For any prefix P of any feasible initial run, there is a feasible finite extension R_t of P with the property that any two R_t-similar runs R and R' provide the same eventual leader $\ell(R') = \ell(R)$.

Proof: Suppose, by way of contradiction, that

$$\forall\text{finite feasible extensions } R_t \text{ of } P: \\ \exists R_t\text{-similar runs } R \text{ and } R' \text{ with } \ell(R') \neq \ell(R),$$

which obviously implies (we drop attributes like "feasible" here for conciseness)

$$\forall R_t \text{ of } P : \forall R = R_t E_t : \exists R_t\text{-similar } R' = R_t E'_t \\ \text{with } \ell(R') \neq \ell(R). \tag{1}$$

This assumption will allow us to prove the existence of a feasible run $\hat{R}$ where the leader changes infinitely often, which contradicts the properties of Ω. Hence, there must indeed be some finite prefix R_t such that any R_t-similar run R' provides $\ell(R') = \ell(R)$ as asserted.

We find it instructive to introduce the essence of our proof by an inductive construction of a sequence of prefixes R_k, $k \geq 1$, of feasible runs, such that R_k is a proper prefix of R_{k+1} and the leader has changed. In the limit, this leads to a feasible run $\hat{R} = \lim_{k\to\infty} R_k$ where the leader changes infinitely often. Since we are dealing with infinite[2] executions, however, we will employ König's infinity lemma [12, Lemma 8.1.2] to establish the existence of $\hat{R}$.

For the induction basis $k = 1$, let R_1 be some prefix of length at least $|P|$ of the initial feasible run where some leader ℓ_1 has already been elected by all processes. Clearly, P is prefix of R_1.

For the induction step, assume that we have already constructed the prefix R_k of some feasible run R, where the leader ℓ_k has been chosen. By (1), there is an R_k-similar (and hence feasible) run R' where some different leader $\ell(R') \neq \ell_k$ is chosen. Let R_{k+1} be a prefix of R' with $|R_{k+1} \setminus R_k| \geq 1$, to guarantee $\lim_{k\to\infty} |R_k| = \infty$, where some[3] new leader ℓ_{k+1} has already been chosen, every correct processor has performed at least one step, and all messages from R_k have been delivered. Since $\ell(R') \neq \ell(R)$, such a prefix R_{k+1} must of course exist. Obviously, R_k is a proper prefix of R_{k+1} and the leader has changed.

For the rigorous proof, based on König's infinity lemma, we define an infinite sequence of non-empty finite sets $V_0, V_1, \dots$ as follows: Consider some arbitrary feasible run R and let $(\ell_1(R), \ell_2(R), \dots)$ with $\ell_{k-1}(R) \neq \ell_k(R)$ for any $k \geq 1$ be a subsequence of the different unique leaders chosen in R, defined inductively as follows: Given $\ell_{k-1}(R)$ along with the prefix R_{k-1} of R at the end of which $\ell_{k-1}(R)$ has been chosen, we define R_k to be the prefix of R [that extends R_{k-1}] that lets (1) every correct processor take at least one step in $R_k \setminus R_{k-1}$, (2) all messages sent in R_{k-1} be delivered in R_k, and (3) another leader $\ell_k(R) \neq \ell_{k-1}(R)$ be chosen at the end of R_k. Moreover, we define $\ell_0(R) = \perp$ and $R_0(R) = P$ for every R.

The sets V_k are defined as follows:

$$\begin{aligned} V_0 &= \{\perp\} \\ V_k &= \{(\ell_1, \dots, \ell_k) | \exists \text{ feasible run } R \\ &\qquad \text{where } \forall 1 \leq i \leq k : \ell_i = \ell_i(R)\} \end{aligned}$$

Because of (1), it follows that $|V_k| \geq 1$ for every $k \geq 0$, and since there are at most n processes, $|V_k| \leq n^k < \infty$.

Moreover, there is a natural tree connecting the elements of neighboring sets V_{k-1}, V_k: There is an edge (v_{k-1}, v_k) for $v_{k-1} \in V_{k-1}$ and $v_k \in V_k$ iff $v_k = (v_{k-1}, \ell)$ for some leader ℓ. Any element in V_1 is connected by an edge to the single element $\perp \in V_0$. Note that the existence of an edge implies that there is a feasible run R where $(\ell_1(R), \dots, \ell_{k-1}(R)) = (\ell_1, \dots, \ell_{k-1}) = v_{k-1}$ and $(\ell_1(R), \dots, \ell_k(R)) = (\ell_1, \dots, \ell_k) = v_k$ where $\ell_{k-1} \neq \ell_k$ for any $k \geq 1$.

It is obvious from our construction that every element in V_k has an edge to exactly one element in V_{k-1}. We can hence apply König's infinity lemma, which asserts an infinite path—and hence a corresponding infinite feasible run $\hat{R}$—in the tree:

König's infinity lemma [12, Lemma 8.1.2]: Let $V_0, V_1, \dots$ be an infinite sequence of disjoint non-empty finite sets, and let G be a graph on their union. Assume that every vertex v in a set V_n with $n \geq 1$ has a neighbour $f(v)$ in V_{n-1}. Then G contains a ray (an infinite path) $v_0, v_1, \dots$ with $v_n \in V_n$ for all n.

Since the leader changes with every k along that path, it changes infinitely often in $\hat{R}$. □

Bear in mind that R_t guaranteed by Lemma 8 is such that all process crashes and all $\diamond$moving-f-source status changes must have occurred already in R_t. Remember also that it is not necessarily the case that all messages sent in R_t are received within R_t.

With a similar proof as for Lemma 8, we can show the following stabilization property w.r.t. effective communication graphs:

Lemma 9: For any prefix P of some feasible initial run, there is a feasible finite extension R_t of P and a feasible run $R = R_t E_t$, with the property that any R_t-similar run R' provides $G_e(R') \subseteq G_e(R)$.

Proof: Suppose, by way of contradiction, that

$$\forall\text{feasible } R : \forall\text{prefixes } R_t : \exists R_t\text{-similar run } R' \\ \text{with } G_e(R') \not\subseteq G_e(R). \tag{2}$$

If we denote by $U_t = \bigcup_{R'} G_e(R')$ the union of the effective communication graphs of all R_t-similar runs R' of R (including R), which is the graph containing the links of all $G_e(R')$, this condition implies

$$\forall\text{feasible runs } R : \forall\text{prefixes } R_t : U_t \not\subseteq G_e(R). \tag{3}$$

We will now derive a contradiction: Our assumption will allow us to inductively construct an infinite sequence of prefixes R_k, $k \geq 1$, of feasible runs where, from

2. Induction provides assertions for finite executions only, which is not sufficient here. In fact, a finite sequence of such prefixes could always be constructed by crashing the eventual leader or changing the status of processes w.r.t. being a moving-f-source or not.

3. Note carefully that ℓ_{k+1} may be different from the eventually chosen leader $\ell(R')$ here, since agreeing on the latter might require all messages (including all slow ones) to be received. By allowing $\ell_{k+1} \neq \ell(R')$, this is not required here.

some finite index k' on, all sets $U_k = U_{|R_k|}$, $k \geq k'$, are the same and all links in this set are used infinitely often. Hence, for any feasible run with prefix $R_{k'}$, in particular, for $\hat{R} = \lim_{k\to\infty} R_k$, all those links must be contained in $G_e(\hat{R})$, which contradicts $U_{k'} \not\subseteq G_e(\hat{R})$ in (3).

For the induction basis $k = 1$, we choose $R_1 = P$, which is a prefix of the initial feasible run. For the induction step, we assume that the prefix R_k of some feasible run has already been constructed. To obtain R_{k+1}, we extend R_k by some prefix of length ≥ 1, to guarantee $\lim_{k\to\infty} |R_k| = \infty$, of some R_k-similar (and hence feasible) run R′ specified in detail below. Obviously, R_k is a proper prefix of R_{k+1}, hence either (a) $U_{k+1} \subset U_k$ or (b) $U_{k+1} = U_k$: After all, the resulting set of R_{k+1}-similar runs is just the set of R_k-similar runs without all runs $\mathcal{Y}_k$ that have prefix R_k but not R_{k+1}. Consequently, U_{k+1} is just U_k without $G_e(Y)$ for all $Y \in \mathcal{Y}_k$.

Since there are only finitely many different effective communication graphs that could be dropped via case (a), there must be some finite index k' such that $U_k = U_{k+1} = U$ for all $k \geq k'$, which leaves us with (b) as the only interesting case. For every link $(p, q) \in U_k$, there is some R_k-similar run R' with $(p, q) \in G_e(R')$ by the definition of U_k. To construct R_{k+1}, we choose a non-zero length prefix of $E_k = R' \setminus R_k$ such that at least one message is sent along (p, q) in E_k and all messages sent in R_{k-1} have been delivered by the end of E_k. Since a different link $(p, q) \in U$ is chosen for every k, such that all links in U are used at least once in round-robin order in $R_{k'}$, $R_{k'+1}$, …, it follows that all links in U are used infinitely often (and, hence, all correct processes take infinitely many steps) in the limiting run.

Finally, we use König's infinity lemma to prove the existence of a feasible limiting run $\hat{R} = \lim_{k\to\infty} R_k$. More specifically, we will start out from the prefix $R_{k'}$ established above, i.e., restrict our attention to $R_{k'}$-similar runs that satisfy $U_k = U = \{1, \ldots, u\}$ for all $k \geq k'$. We classify those runs into sets V_k according to the sequence of chosen communication links $\in U$.

Consider some arbitrary feasible extension E of $R_{k'}$ and let $L(E) = (l_1(E), l_2(E), \ldots)$ be a subsequence of the links chosen for sending a message in E, with broadcast messages resolved in an arbitrary order, defined as follows: Given $l_{k-1}(E)$ along with the prefix E_{k-1} of E at the end of which $l_{k-1}(E)$ is used for sending a message, we define E_k to be the prefix of E [that extends E_{k-1}] where (1) all messages sent in E_{k-1} are delivered in E_k, and (2) at the end of which the link $l_k(E) \neq l_{k-1}(E)$ is used for sending a message. Moreover, we define $l_0(R) = \perp$ and $E_0(R) = \{\}$ for every E. The actual sequence of links $l_1(E), l_2(E), \ldots$ is chosen in an arbitrary but fixed round-robin sequence $S = (1, 2, \ldots, u, 1, 2, \ldots)$ of the links in U. Let S_k be the prefix of the first k elements of S, with $S_0 = (\perp)$.

We classify feasible extensions into sets V_k according to the maximum S_k that appears as a (not necessarily consecutive) subsequence in $L(E)$. More specifically, V_k will contain the element S_k if there is some feasible extension E the sequence of chosen links $L(E)$ of which contains all links in S_k in the appropriate order. In addition, if there is a feasible extension E the sequence of chosen links of which has length at least k and contains S_{k-1} but not S_k, then V_k contains the element $(S_{k-1}, \perp)$.

Formally, the sets V_k are hence defined as follows:

$$
\begin{aligned}
V_0 &= \{(\perp)\} \\
V_k &= \{S_k | \exists \text{ extension } E \\
&\qquad \text{where } l_{\pi(i)}(E) - 1 \equiv i - 1 \mod u \\
&\qquad \text{for } 1 \leq i \leq k \text{ and } \pi(i) < \pi(i+1)\} \cup \\
&\quad \{(S_{k-1}, \perp) | \exists \text{ extension } E \text{ with } |L(E)| \geq k \\
&\qquad \text{where } l_{\pi(i)}(E) - 1 \equiv i - 1 \mod u \\
&\qquad \text{for } 1 \leq i \leq k-1 \text{ and } i \leq \pi(i) < \pi(i+1) \\
&\qquad \text{but not for } i = k\}
\end{aligned}
$$

Clearly, $|V_k| \leq 2 < \infty$. Moreover, as explained in the inductive construction above, we can guarantee that $S_k \in V_k$ and hence $|V_k| \geq 1$ for every k. As in Lemma 8, there is again a natural tree connecting the elements of neighboring sets V_{k-1}, V_k: Both elements $S_k \in V_k$ and $(S_{k-1}, \perp) \in V_k$, if present, have an edge to the element $S_{k-1} \in V_{k-1}$. We can hence apply König's infinity lemma [12, Lemma 8.1.2], which asserts an infinite path in the tree and hence the existence of a feasible extension $\hat{E}$—and thus a feasible run $\hat{R} = R_{k'}\hat{E}$—where all links in U are used infinitely often. It hence follows by the definition of the effective communication graph $G_e(\hat{R})$ of run $\hat{R}$ that $U = U_k \subseteq G_e(\hat{R})$ for any $k \geq k'$. Since every R_k is of course a prefix of $\hat{R}$, this contradicts (3), however. Hence, there must indeed be some finite prefix $R_t = R_{k'}$ and some run R such that any R_t-similar extension R' provides $G_e(R') \subseteq G_e(R)$ as asserted. □

Using Lemma 8 and 9, we can now prove that the eventual leader must have all $n-1$ outgoing links in the eventual communication graph $G_e(R)$ in some feasible run R in $\mathcal{S}^{\rightarrow}_{f*}$. This contrasts with the lower bound of only f outgoing links for system $\mathcal{S}^{\rightarrow}_{f}$ with an $\diamond f$-source [5], and must be considered as the price for moving timely links.

Theorem 3: For all $n > f + 1 \geq 2$, in a system $\mathcal{S}^{\rightarrow}_{f*}$ with reliable links and n processes where up to f processes may crash, the eventual leader must have all $n - 1$ outgoing links in the eventual effective communication graph in some run. This holds even when every process is a perpetual moving-f-source, and δ is known.

Proof: Given any prefix P of some feasible failure-free run, consider the feasible run $R8$ and the finite prefix $R8_t$ guaranteed by Lemma 8. We can choose the initial run resp. the prefix P in the application of Lemma 9 to be just $R8$ resp. $R8_t$. Recall that all crashes, if any, must have occurred in $R8_t$ already. As a result, we are assured about the existence of some feasible run R and a finite prefix R_t, where for every R_t-similar run R' not only $G_e(R') \subseteq G_e(R) = G_e$ but also $\ell(R') = \ell(R) = \ell$. The

latter holds since a run R' that is R_t-similar is obviously also $R8_t$-similar if $R8_t$ is a prefix of R_t.

Suppose that there is at least one link from ℓ to some process p missing in G_e. Consider two R_t-similar runs R^1 and R^2, where in both extensions (after R_t) process ℓ could send timely over this missing link to p (recall that an unused link can be considered timely), as well as to $f-1$ other processes. Denote the set of these $f-1$ processes and ℓ as A, and let $B = \Pi - A$ with $p \in B$ be the remaining set of $n - f \geq 2$ processes. Every process in B sends timely to all f processes in A and arbitrarily to all other processes in B, and every process in A sends timely to the $f-1$ other processes in A. In addition, every process in A also sends timely to process p in R^1, and to some process $q \in B$ with $q \neq p$ in R^2. Finally, every process in A has $n - f - 1$ slow outgoing links to the remaining processes in B as well, which will experience some large delay $\Delta + 1$ as explained below. Due to our construction, all processes could be a moving-f-source in both runs.

Now consider runs $R^{crash1} = R^{crash2} = R^{crash}$ where all processes in A (which include ℓ) crash at t, i.e., at the end of R_t. Clearly, a new leader must be elected within some extended prefix $R^{crash}_{t+\Delta}$. Note carefully that $t + \Delta$ may be such that even all slow messages from the surviving processes (in B) must have arrived, i.e., are required for electing the new leader. Processes p and q now face a dilemma: If we choose the delay of slow messages sent by processes $\in A$ in R^1 and R^2 after R_t to be $\Delta + 1$, $R^{crash}_{t+\Delta}$ is indistinguishable from $R^1_{t+\Delta}$ for q. Similarly, $R^{crash}_{t+\Delta}$ is indistinguishable from $R^2_{t+\Delta}$ for p. Thus, if p resp. q demotes ℓ, it behaves illegal w.r.t. R^2 resp. R^1. If those processes don't demote ℓ, they behave illegal w.r.t. R^{crash}.

Hence, the implementation of Ω cannot be correct, such that all $n-1$ outgoing links from ℓ must indeed be in G_e as asserted. After all, $R^{crash}_{t+\Delta}$ would no longer be indistinguishable from $R^1_{t+\Delta}$ for q if the link from ℓ to p really carried a timely message, instead of being vacuously timely because the link is not used. □

Finally, we will provide our major lower bound theorem. It shows that the eventual communication graph $G_e(R)$ in some feasible run R must contain as many as $\Omega(nf)$ links in $\mathcal{S}^{\rightarrow}_{f*}$. Again, this sharply contrasts with the lower bound of only f outgoing links for system $\mathcal{S}^{\rightarrow}_{f}$ with an $\diamond f$-source [5], and must also be considered as the price for moving timely links.

Theorem 4: For all $n > f + 1 \geq 2$, in a system $\mathcal{S}^{\rightarrow}_{f*}$ with reliable links and n processes where up to f processes may crash, any implementation of Ω requires at least $\frac{nf}{2}$ links to carry messages forever in some run. This holds even when every process is a perpetual moving-f-source, and δ is known.

Proof: Given any prefix P of some feasible failure-free run, consider the feasible run $R8$ and the finite prefix $R8_t$ guaranteed by Lemma 8. We can choose the initial run resp. the prefix P in the application of Lemma 9 to be just $R8$ resp. $R8_t$. As a result, we are assured about the existence of some feasible run R and a finite prefix R_t, where for every R_t-similar run R' not only $G_e(R') \subseteq G_e(R) = G_e$ but also $\ell(R') = \ell(R) = \ell$. The latter holds since a run R' that is R_t-similar is obviously also $R8_t$-similar if $R8_t$ is a prefix of R_t.

Let $\deg^{in}_p$ resp. $\deg^{out}_p$ be the in-degree resp. out-degree of process p in G_e. Assume, by way of contradiction, that there is at least one process p with $\deg^{in}_p + \deg^{out}_p \leq f-1$ in G_e. Since we know already from Theorem 3 that ℓ has $\deg^{out} = n-1$ in G_e, we can assume $p \neq \ell$. Let X^{in} resp. X^{out} be the sets of processes with links to resp. from p. Further let Y be an arbitrary set of processes such that $|X^{in} \cup X^{out} \cup Y| = f$, $\ell \in Y$, and $p \notin Y$. Let $A = X^{in} \cup X^{out} \cup Y$ and $B = \Pi - A - \{p\} \neq \emptyset$.

Now consider two R_t-similar runs R^1 and R^2, where in both extensions (after R_t) process p could send timely to all processes in A (although only processes in $X^{out} \subset A$ receive those messages) and processes in A and in B could send timely to all processes in A. However, R^1 and R^2 differ in that all processes in A could also send timely to p in R^1 (although p receives messages from $X^{in} \subset A$ only), whereas in R^2 all processes in A could send timely to some process $q \in B$ instead. Note that every process in A has $n - f - 1$ slow outgoing links as well, which will experience some large delay $\Delta + 1$ as explained below. By our construction, all processes could be a moving-f-source in both runs. However, process p and all processes in B are totally partitioned from each other in G_e if the processes in A were removed.

Now consider runs $R^{crash1} = R^{crash2} = R^{crash}$ where all processes in A (which include ℓ) crash at t, i.e., at the end of R_t. Clearly, a new leader must be elected within some extended prefix $R^{crash}_{t+\Delta}$. Note carefully that $t + \Delta$ may be such that even slow messages from p must have arrived, i.e., are required for electing the new leader. Processes p and B now face a dilemma: If we choose the delay of slow messages sent by processes $\in A$ in R^1 and R^2 after R_t to be $\Delta + 1$, $R^{crash}_{t+\Delta}$ is indistinguishable from $R^1_{t+\Delta}$ for all processes in B. Similarly, $R^{crash}_{t+\Delta}$ is indistinguishable from $R^2_{t+\Delta}$ for p. Thus, if p resp. some process in B demotes ℓ or uses a link outside G_e, it behaves illegal w.r.t. R^2 resp. R^1. If those processes don't demote ℓ and/or do not use a link outside G_e, they behave illegal w.r.t. R^{crash}. Hence, the implementation of Ω cannot be correct. Consequently, every correct algorithm has $\deg^{in}_p + \deg^{out}_p \geq f$ for every process p. The number of links that carry messages forever is hence $|\Lambda| = \frac{1}{2}\sum_{p\in\Pi} \deg^{in}_p + \deg^{out}_p \geq \frac{nf}{2}$. □

The following result follows immediately from the fact that Theorem 3 and 4 have been established for a stronger model:

Corollary 5: For $n > f + 1 \geq 2$, any implementation of Ω in $\mathcal{S}^{\rightarrow}_{f*}$ requires at least $\Omega(nf)$ links, including all $n-1$ outgoing links from the eventual leader, to carry messages forever in some run.

4.2 Attaining the lower bound

The lower bounds established in Corollary 5 show that any Ω implementation in $\mathcal{S}^{\rightarrow}_{f*}$ has high communication costs. In this section, we present an algorithm (Algorithm 2) that attains this lower bound, i.e., uses only $O(nf)$ links forever.

In view of such high costs, one may wonder whether there is an algorithm that works correctly in $\mathcal{S}^{\rightarrow}_{f*}$, but automatically reduces its communication demand when the system stabilizes from $\mathcal{S}^{\rightarrow}_{f*}$ to a system with stronger synchrony properties, at some unknown point in time. Interestingly, Algorithm 2 also meets this requirement: Under the additional condition that links are reliable and that there exists some time t, after which some process p remains the leader forever and is never suspected by any live process[4], the communication complexity reduces from $O(nf)$ to $O(n)$ some finite time after t.

Algorithm 2 employs echoing (= rebroadcasting) of all messages. In order not to complicate the code further with messages counters and retransmissions, we assume reliable message delivery of messages from correct processes. To ensure that the algorithm matches the $\Omega(nf)$ quiescence lower bound established in Corollary 5, we also assume that the SUSPECT messages (and their echoes) are disseminated to all processes via flooding over some (fixed) $(f+1)$-node-connected and $(f+1)$-regular overlay topology. The appropriate protocol is also omitted from the code for simplicity.

To bring down the communication complexity from $O(n^2)$ to $O(nf)$ resp. $O(n)$, a process should send ALIVE messages only when it considers itself the leader. Although a process not considering itself leader no longer sends ALIVE messages, its counter should not increase on other processes due to suspicions. To this end, each process maintains a new boolean status vector $enabled$, with one bit for each other process. Whenever the suspicion counter p holds for q increases, p sets $enabled_p[q]$ to false and waits for q to also learn that its own counter has advanced. Subsequently, $enabled_p[q]$ is set to true only upon receipt of an ALIVE message with the current or higher counter from q. The entry becomes *disabled* (set to `false`) if q explicitly sends a DISABLE message for this counter. This guarantees that the suspicion counter of a $\diamond$moving-f-source does not increase after it becomes timely.

Because messages are echoed and may arrive out of order, a DISABLE message is disregarded if it arrives after an ALIVE message with the same (or higher) counter is received. To this end, a process records in the $maxenabled$ vector a highest enabled counter value for each process. With the $enabled$ vector, a process p sends SUSPECT messages only on an enabled process when the corresponding timer expires.

The status vector also addresses the following problem: A process q that has not been a leader for a while may become leader at some point. The problem is that different processes may learn that q is the new leader at different times. Since q does not send ALIVE messages until it believes itself to be the leader, other processes may start suspecting it even before the first ALIVE message goes out. In the case of a new leader q as described before, the $enabled$ flag for q is `false` in its current counter until the first ALIVE message from q arrives. Therefore, no premature suspicions are generated. We proceed with a proof of correctness.

4. In other words, we consider runs of Ω implementations in $\mathcal{S}^{\rightarrow}_{n-1}$ in which an $\diamond(n-1)$-source (and not only a $\diamond$moving-f-source) exists.

Algorithm 2 Ω with reduced communication, code for a process p

Variables
1: $\forall q \in \Pi$: $counter_p[q] \in \mathbb{N}$, initially 0
2: $\forall q \in \Pi$: $enabled_p[q] \in \{\texttt{true}, \texttt{false}\}$, initially `false`
3: $\forall q \in \Pi$: $maxenabled_p[q] \in \mathbb{N}$, initially 0
4: $\forall q \in \Pi\ \forall s \in \mathbb{N}$: $suspect_p[q][s] \subseteq \Pi$, initially $\emptyset$
5: $\forall q \in \Pi\ \forall s \in \mathbb{N}$: $reported_p[q][s] \in \{\texttt{true}, \texttt{false}\}$, initially `false`
6: $seq_p, rseq_p \in \mathbb{N}$, initially 0
7: Δ_p, initially η

Initially
8: start Tasks 1,2
9: start $timer$ with Δ_p

Task 1
10: **every** η timesteps **do**
11: $prevleader_p \leftarrow leader_p$
12: $leader_p \leftarrow q$, where $\langle counter_p[q], q\rangle$ is
13: minimal among $\{q \mid enabled_p[q] = \texttt{true} \vee q = p\}$
14: **if** $leader_p = p$ **then**
15: send $\langle$ALIVE$, p, seq_p, counter_p\rangle$ to all $q \in \Pi$
16: **else if** $prevleader_p = p$ **then**
17: send $\langle$DISABLE$, p, counter_p[p]\rangle$ to all $q \in \Pi$
18: $seq_p \leftarrow seq_p + 1$

Task 2
19: **upon** receive $\langle$ALIVE$, q, s, c\rangle$ **do**
20: $reported_p[q][s] \leftarrow \texttt{true}$
21: **for all** $r \in \Pi$ **do**
22: **if** $c[r] > counter_p[r]$ **then**
23: $counter_p[r] \leftarrow c[r]$
24: $enabled_p[r] \leftarrow \texttt{false}$
25: **if** $c[q] = counter_p[q]$ **then**
26: $enabled_p[q] \leftarrow \texttt{true}$
27: $maxenabled_p[q] \leftarrow c[q]$
28: **if** any change done to $counter_p$ or to $enabled_p$ **then**
29: send $\langle$ALIVE$, q, s, c\rangle$ to all $r \in \Pi$ /* echo */

30: **upon** receive $\langle$DISABLE$, q, c\rangle$ **do**
31: **if** $q \neq p$ **then**
32: send $\langle$DISABLE$, q, c\rangle$ to all $r \in \Pi$ /* echo */
33: **if** $counter_p[q] \leq c$ **then**
34: $counter_p[q] \leftarrow c$
35: **if** $maxenabled_p[q] < c$ **then**
36: $enabled_p[q] \leftarrow \texttt{false}$

37: **upon** expiration of $timer$ **do**
38: **for all** q, where $enabled_p[q] = \texttt{true}$ and $reported_p[q][rseq_p] = false$ **do**
39: send $\langle$SUSPECT$, q, p, rseq_p\rangle$ to all $r \in \Pi$
40: $reported_p[q][rseq_p] \leftarrow \texttt{true}$
41: $rseq_p \leftarrow rseq_p + 1$
42: $\Delta_p \leftarrow \Delta_p + 1$
43: set $timer$ to Δ_p

44: **upon** receive $\langle$SUSPECT$, q, r, s\rangle$ **do**
45: **if** $q \neq p$ **then**
46: send $\langle$SUSPECT$, q, r, s\rangle$ to all $r \in \Pi$ /* echo */
47: $suspect_p[q][s] \leftarrow suspect_p[q][s] \cup \{r\}$
48: **if** $|suspect_p[q][s]| \geq n - f$ **then**
49: $counter_p[q] \leftarrow counter_p[q] + 1$

Lemma 10: For every $\diamond$moving-f-source process p there exists a bound b_p such that for all $q \in \Pi$, always $counter_q[p] \leq b_p$.

Proof: Let t be a time after which p is a moving-f-source. Consider a time $t' \geq t$ after which any DISABLE message from p that was sent prior to t has been delivered by all correct processes. After time t', there may be no suspicion messages generated due to late arrival of disabling message from p. Hence, after that time, there is no sequence number for which $n-f$ processes suspect p. It easily follows that suspicion counters on p never increase beyond $b_p = \max_{q\in\Pi}\{counter_q[p]\}$ at time t'. □

Lemma 11: If for some correct process q at any point $counter_q[p] = c$, then eventually for all correct processes r it holds that $counter_r[p] \geq c$.

Proof: This follows immediately from the fact that SUSPECT messages are echoed. □

Lemma 12: If a process q has crashed then eventually for all correct processes p it forever holds that $enabled_p[q] = \texttt{false}$.

Proof: Due to Lemma 11, the highest enabled counter value for q held by any correct process is eventually held by all. If any correct process receives a DISABLE message from q with this counter value, then it is echoed to all. Otherwise, since q crashes, after one more timer expiration, there are $n-f$ suspicions with this counter value for sure. Therefore, eventually all processes have the same, highest counter value, which is never enabled since q has crashed. □

Theorem 5: Algorithm 2 maintains Ω.

Proof: By Lemma 10 and Lemma 12, there exists a non-crashed process p whose counter is bounded and forever has the minimal value among all bounded counters. By Lemma 11, eventually p knows that all other counters are either higher than its own, or are disabled. Therefore, p enables this counter. Therefore, every other correct process q eventually has $enabled_q[p] = \texttt{true}$ and $counter_q[p]$ fixed. Furthermore, like p, q also eventually has all other counter values higher than p's or disabled. Hence, p is the leader. □

Theorem 6: Eventually, Algorithm 2 uses only $O(nf)$ communication channels infinitely often.

Proof: By Theorem 5, eventually all non-crashed processes consider p to be the leader. Hence, only process p continues sending ALIVE messages on $n-1$ links forever. All SUSPECT messages are exchanged via the $(f+1)$-node-connected and $(f+1)$-regular overlay. Thus, in the effective communication graph, every process has $f+1$ outgoing links, except the leader, which has $n-1$ outgoing links, resulting in a total of $(f+2)(n-1) = O(nf)$ non-quiescent links. □

Theorem 7: In a system where the leader stabilizes from a moving-f-source to an $(n-1)$-source, Algorithm 2 uses only $n-1$ communication channels infinitely often.

Proof: By Theorem 5, eventually all non-crashed processes consider p to be the leader. Hence, only process p continues sending ALIVE messages forever. By assumption, eventually there are no suspicions of the leader, so no SUSPECT messages are sent after a certain time in the execution. □

Algorithm 3 Ω Implementation with eventually stabilizing timeouts, code for a process p

Variables
1: $\forall q \in \Pi$: $counter_p[q] \in \mathbb{N}$, initially η
2: $\forall q \in \Pi\ \forall s \in \mathbb{N}$: $suspect_p[q][s] \subseteq \Pi$, initially $\emptyset$
3: $\forall q \in \Pi\ \forall s \in \mathbb{N}$: $reported_p[q][s] \in \{\texttt{true}, \texttt{false}\}$, initially $\texttt{false}$
4: $seq_p, rseq_p \in \mathbb{N}$, initially 0

Initially
5: start Tasks 1,2
6: $\forall q \in \Pi$: start $timer(q)$ with $counter_p[q]$

Task 1
7: **every** η timesteps **do**
8: send $\langle$ALIVE$, p, seq_p, counter_p\rangle$ to all $q \in \Pi$
9: $seq_p \leftarrow seq_p + 1$
10: $leader_p \leftarrow q$, where $\langle counter_p[q], q\rangle$ is minimal

Task 2
11: **upon** receive $\langle$ALIVE$, q, s, c\rangle$ **do**
12: $reported_p[q][s] \leftarrow \texttt{true}$
13: **for all** $r \in \Pi$ **do**
14: $counter_p[r] \leftarrow \max\{counter_p[r], c[r]\}$

15: **upon** expiration of $timer(q)$ **do**
16: **for all** q, where $reported_p[q][rseq_p] = false$ **do**
17: send $\langle$SUSPECT$, q, p, rseq_p\rangle$ to all $r \in \Pi$
18: $reported_p[q][rseq_p] \leftarrow \texttt{true}$
19: $rseq_p \leftarrow rseq_p + 1$
20: set $timer(q)$ to $counter_p[q]$

21: **upon** receive $\langle$SUSPECT$, q, r, s\rangle$ **do**
22: $suspect_p[q][s] \leftarrow suspect_p[q][s] \cup \{r\}$
23: **if** $|suspect_p[q][s]| \geq n - f$ **then**
24: $counter_p[q] \leftarrow counter_p[q] + 1$

Theorem 3 reveals that Algorithm 2 is also optimal in this respect: Since the algorithm cannot know whether a moving-f-source has indeed become an $(n-1)$-source forever, or just mimics this for some finite time, the leader must always use $n-1$ links infinitely often. Note that this argument is not contradicted by the communication-optimal algorithm from [5], which uses only f links infinitely often, since the latter does not work in $\mathcal{S}^{\rightarrow}_{f*}$.

5 Ω IMPLEMENTATION IN $\mathcal{S}^{\rightarrow}_{f*}$ WITH EVENTUALLY STABILIZING TIMEOUTS

Algorithms 1 and 2 suffer from ever increasing timeouts. In this section, we provide an implementation of Ω where the timeout values eventually stabilize for timely links. The key idea is to use the $counter_p[q]$ variable as the timeout value for messages from q. This works, since $counter_p[q]$ increases until at least f messages from a $\diamond$moving-f-source q are timely with respect to $counter_p[q]$—exactly what we need for an adaptive timeout value.

The proof of correctness is similar to the proof of the simple algorithm in Section 3.

Lemma 13: If p is correct and q is faulty then $counter_p[q]$ is unbounded.

Proof: If q is faulty then eventually it stops sending $\langle \text{ALIVE}, p, seq, c\rangle$ messages. Therefore there is a sequence number s, such that for all $s' \geq s$ and every correct process p', $reported_{p'}[q][s']$ is never set `true` in line 12, and thus p' sends a $\langle \text{SUSPECT}, q, p', s'\rangle$ to all processes (line 17). Since there are at least $n-f$ correct processes, it follows that p increments $counter_p[q]$, and since there are infinitely many s', this happens infinitely often. □

Lemma 14: Let σ_p^k denotes the time where p sends $\langle \text{ALIVE}, p, k, *\rangle$ and ρ_q^k be the time where q timeouts this message, i.e., increments the receiver sequence number $rseq_q$ from k to $k+1$. If $counter_q[p]$ grows without bound [in fact, becomes larger than Φ], then $\exists k' : \rho_q^k \geq \sigma_p^k + \delta$ for all $k \geq k'$.

Proof: Let k_0 be the value of the receiver sequence number $rseq_q$ when $counter_q[p]$ has just reached $(\Phi+1)\eta$, and $T = \sigma_p^{k_0} - \rho_q^{k_0}$. Note that T may also be negative. For all $k > k_0$, $\sigma_p^k \leq \sigma_p^{k_0} + (k-k_0)\Phi\eta$, whereas $\rho_q^k = \rho_q^{k-1} + counter_q[p] \geq \rho_q^{k-1} + (\Phi+1)\eta$ and hence $\rho_q^k \geq \rho_q^{k_0} + (k-k_0)(\Phi+1)\eta$. Hence, $\sigma_p^k - \rho_q^k \leq \sigma_p^{k_0} + (k-k_0)\Phi\eta - \rho_q^{k_0} - (k-k_0)(\Phi+1)\eta \leq T - (k-k_0)\eta$. Hence, there is some k' such that indeed $\rho_q^k \geq \sigma_p^k + \delta$ for all $k \geq k'$ as asserted. □

Lemma 15: If p is a correct $\diamond$moving-f-source then, for every process q, $counter_q[p]$ is bounded.

Proof: By way of contradiction, assume that $counter_q[p]$ is unbounded. Then there is at least one process r, where line 23 is true infinitely often for p, which requires $n-f$ $\langle \text{SUSPECT}, p, *, s\rangle$ for infinitely many s. Since p is a $\diamond$moving-f-source, there is a sequence number s_0, so that for every $s \geq s_0$, there is a set $Q(t_s)$ of at least f processes that receive $\langle \text{ALIVE}, p, s, c\rangle$ by time δ after it was sent at time t_s. Since p never suspects itself, there must be at least one process $r_s \in Q(t_s)$ that sends $\langle \text{SUSPECT}, p, *, s\rangle$ in order to reach the $n-f$ quorum, despite of the fact that $\langle \text{ALIVE}, p, s, *\rangle$ was received timely at r_s, for infinitely many s. Since there are only finitely many processes in the system, there must be at least one correct process r that occurs infinitely often among the processes r_s. This process must hence infinitely often time out a timely message. However, since by assumption $counter_q[p]$ is unbounded, and q uses $counter_q[p]$ as timeout for messages from q, eventually the timeout will be larger than any transmission delay plus the desynchronization of p and q, which is bounded by Lemma 14. □

Now, the Lemmas 2, 6, and 7 also hold for Algorithm 3. Thus, we also have:

Theorem 8: Algorithm 3 implements Ω in system $\mathcal{S}^{\rightarrow}_{f*}$

6 Solving Consensus in $\mathcal{S}^{\rightarrow}_{f*}$

It is well known, that consensus [16] can be solved in system with Ω and reliable [9], [10] or fair-lossy [2], [3] links, if $n > 2f$.

Corollary 6: For $n > 2f$, consensus can be solved in systems $\mathcal{S}^{u\rightarrow}_{f*}$ and $\mathcal{S}^{b\rightarrow}_{f*}$.

7 Conclusions

We presented a new system model, which we believe to be the weakest model for implementating Ω and thus consensus proposed so far: It assumes just a single process with f outgoing moving eventually timely links; all other links can be totally asynchronous. Our tight lower bounds regarding the communication complexity show, however, that those weak timing assumptions have a price: For example, rather than using only f links forever, $\Omega(nf)$ links must be used forever in our moving model. Our results thus reveal an interesting tradeoff between synchrony assumptions and communication complexity.

Acknowledgements

We like to thank Sergio Rajsbaum, Corentin Travers, Hugues Fauconnier, Martin Biely and Josef Widder for their valueable feedback on an earlier version of our paper.

References

[1] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing Omega with weak reliability and synchrony assumptions. In *Proceeding of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC'03)*, pages 306–314, New York, NY, USA, 2003. ACM Press.

[2] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *Workshop on Distributed Algorithms*, pages 126–140, 1997.

[3] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science*, 220(1):3–30, June 1999.

[4] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election. In *DISC '01: Proceedings of the 15th International Conference on Distributed Computing*, pages 108–122. Springer-Verlag, 2001.

[5] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 328–337, St. John's, Newfoundland, Canada, 2004. ACM Press.

[6] Emmanuelle Anceaume, Antonio Fernández, Achour Mostéfaoui, Gil Neiger, and Michel Raynal. A necessary and sufficient condition for transforming limited accuracy failure detectors. *J. Comput. Syst. Sci.*, 68(1):123–133, 2004.

[7] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM (JACM)*, 41(1):122–152, 1994.

[8] Anindya Basu, Bernadette Charron-Bost, and Sam Toueg. Crash failures vs. crash + link failures. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, page 246, Philadelphia, Pennsylvania, United States, 1996. ACM Press.

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

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

[11] Francis C. Chu. Reducing Ω to $\diamond W$. *Information Processing Letters*, 67(6):298–293, 1998.

[12] Reinhard Diestel. *Graph Theory (3rd ed.)*. Springer, 2006.

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

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

[15] Christof Fetzer, Ulrich Schmid, and Martin Süßkraut. On the possibility of consensus in asynchronous systems with finite average response times. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05)*, pages 271–280, Washington, DC, USA, June 2005. IEEE Computer Society.

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

[17] Rachid Guerraoui and André Schiper. "Γ-accurate" failure detectors. In Özalp Babaoğlu, editor, *Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG'96)*, volume 1151 of *LNCS*, pages 269–286, Berlin / Heidelberg, Oct 1996. Springer Verlag.

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

[19] Gérard Le Lann and Ulrich Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003.

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

[21] Achour Mostéfaoui and Michel Raynal. Solving consensus using chandra-toueg's unreliable failure detectors: A general quorum-based approach. In P. Jayanti, editor, *Distributed Computing: 13th International Symposium (DISC'99)*, volume 1693 of *Lecture Notes in Computer Science*, pages 49–63, Bratislava, Slovak Republic, September 1999. Springer-Verlag GmbH.

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

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

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

[25] Stephen Ponzio and Ray Strong. Semisynchrony and real time. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92)*, pages 120–135, Haifa, Israel, November 1992.

[26] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Proc. 6th Annual Symposium on Theor. Aspects of Computer Science (STACS'89)*, LNCS 349, pages 304–313, Paderborn, Germany, February 1989. Springer-Verlag.

[27] Ulrich Schmid, Bettina Weiss, and John Rushby. Formally verified byzantine agreement in presence of link faults. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 608–616, Vienna, Austria, July 2-5, 2002.

[28] Paul M.B. Vitányi. Distributed elections in an archimedean ring of processors. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 542–547. ACM Press, 1984.

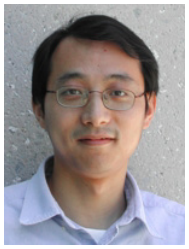
[29] Josef Widder, Gérard Le Lann, and Ulrich Schmid. Failure detection with booting in partially synchronous systems. In *Proceedings of the 5th European Dependable Computing Conference (EDCC-5)*, volume 3463 of *LNCS*, pages 20–37, Budapest, Hungary, April 2005. Springer Verlag.

[30] Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proc. of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC '98)*, pages 297–308, 1998.

Martin Hutle received his Master degree and his PhD in computer science from Vienna University of Technology, in 2002 and 2005, respectively. From 2004-2006 he was working on the European 6th Framework project ASSERT at the Institute of Computer Engineering, Vienna University of Technology. Since July 2006 he has a postdoctoral position at EPFL. His research interests are various topics in fault-tolerant distributed computing.

Dahlia Malkhi

Ulrich Schmid received his diploma (1985) and Dr.techn. degree (1986) from the Technische Universität Wien (computer science and mathematics). He is now full professor and head of the Embedded Computing Systems Group at TU Vienna's Institut für Technische Informatik. Ulrich Schmid authored and co-authored numerous papers in the field of theoretical and technical computer science and received several awards and prices, like the Austrian START-price 1996. He also spent several years in industrial electronics and embedded systems design. 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. Ulrich Schmid is member of IEEE Computer Society and EATCS.



Lidong Zhou Lidong Zhou is a researcher at Microsoft Research Silicon Valley. He obtained his Ph.D. in Computer Science from Cornell University in 2001. His research interests are in distributed systems, security, and computer networks.