Communication-Efficient Leader Election and Consensus with Limited Link Synchrony

[Extended Abstract]

Marcos K. Aguilera* Carole Delporte-Gallet† Hugues Fauconnier‡ Sam Toueg§

ABSTRACT

We study the degree of synchrony required to implement the leader election failure detector Ω and to solve consensus in partially synchronous systems. We show that in a system with n processes and up to f process crashes, one can implement Ω and solve consensus provided there exists some (unknown) correct process with f outgoing links that are eventually timely. In the special case where $f = 1$, an important case in practice, this implies that to implement Ω and solve consensus it is sufficient to have just *one* eventually timely link — all the other links in the system, $\Theta(n^2)$ of them, may be asynchronous. There is no need to know which link $p \rightarrow q$ is eventually timely, when it becomes timely, or what is its bound on message delay. Surprisingly, it is not even required that the source p or destination q of this link be correct: either p or q may actually crash, in which case the link $p \rightarrow q$ is eventually timely in a trivial way, and it is useless for sending messages. We show that these results are in a sense optimal: even if every process has $f - 1$ eventually timely links, neither Ω nor consensus can be solved.

We also give an algorithm that implements Ω in systems where some correct process has f outgoing links that are eventually timely, *such that eventually only f links carry messages*, and we show that this is optimal. For $f = 1$, this algorithm ensures that all the links, except for one, eventually become quiescent.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.5 [**Operating Systems**]: Reliability; F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Algorithms, Design, Reliability, Theory

Keywords

Distributed systems, fault tolerance, asynchronous systems, synchronous systems, partially synchronous systems, failure detector, leader election, consensus

1. INTRODUCTION

Consider a system S of n processes where processes are synchronous and links are reliable, but *one* process may crash. How much link synchrony is necessary to solve consensus in S? From the results in [11, 10, 9], we know that: (a) if *all* the links are *asynchronous*, then consensus cannot be solved, and (b) if *all* the links are *eventually timely* (denoted $\diamond$-*timely*) then consensus can be solved.[1]

*HP Laboratories, 1501 Page Mill Road, Palo Alto, California, 94304, USA, marcos.aguilera@hp.com

†LIAFA, Université D. Diderot, 2 Place Jussieu, 75251, Paris Cedex 05, France, cd@liafa.jussieu.fr

‡LIAFA, Université D. Diderot, 2 Place Jussieu, 75251, Paris Cedex 05, France, hf@liafa.jussieu.fr

§Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, sam@cs.toronto.edu. This author was partially supported by the National Science and Engineering Research Council of Canada, grant 72020311.

PODC'04, July 25–28, 2004, St. John's, Newfoundland, Canada.

[1]Roughly speaking, a link is *eventually timely* if there is some bound δ and a time after which every message sent to a correct process is received within δ units of time. If this bound holds from time 0, we say that the link is *timely*.

In this paper we show that to solve consensus in S, it is sufficient to have just *one*, directed link $p \rightarrow q$ that is $\diamond$-timely — all the other links, $\Theta(n^2)$ of them, can be asynchronous. There is no need to know which link is $\diamond$-timely, when it becomes timely, or what is its bound on message delay. Surprisingly, it is not even required that the source p or destination q of this link be correct: p or q may actually crash, in which case the link $p \rightarrow q$ is $\diamond$-timely in a trivial way, and it is useless for sending messages! Nevertheless, the existence of one such directed link is sufficient for solving consensus in S.

The above result assumes at most one process may crash, i.e., $f = 1$. What happens in the general case $f \geq 1$? We also explore this question, and do so with respect to two related problems: solving consensus, and implementing Ω — a failure detector that can be considered as an *eventual leader elector* [6].

Roughly speaking, with Ω every process p has a local variable $leader_p$ that contains the identity of a single process that p currently trusts to be operational (p considers this process to be its current leader). Initially, different processes may have different leaders, but Ω guarantees that there is a time after which all processes have the *same*, *non-faulty* leader. Failure detector Ω is important for both theoretical and practical reasons: it has been shown to be the weakest failure detector with which one can solve consensus [6], and it is the failure detector used by several consensus algorithms, including some that are used in practice (e.g., [12, 13, 14, 17, 19]).

To state our results, we define the notions of j-source and $\diamond j$-source, as follows. A *$\diamond j$-source* is a process p that has at least j output links $p \rightarrow p_1, p \rightarrow p_2, \ldots, p \rightarrow p_j$ that are *$\diamond$-timely*. It is *not* required that p or any of the $p_1, p_2, \ldots, p_j$ be correct processes. But if p is a correct process, we say that p is a *correct* $\diamond j$-source. A *j-source* is a process p that has at least j output links that are *timely*. A *correct* j-source is a j-source that is a correct process. Note that, by definition, a j-source is also a $(j-1)$-source and a $\diamond j$-source.

Henceforth, we consider systems with n synchronous processes f of which may crash ($n > f \geq 1$). Moreover, to strengthen our possibility results, we assume that links can drop messages but are *fair lossy* [3]: roughly speaking, messages carry a type, and if a process sends an infinite number of messages of a type through a link then the link delivers an infinite number of messages of that type. The question is how much link synchrony is needed to implement Ω or solve consensus in such systems. In this paper, we show the following:

- Ω can be implemented if there is at least one correct $\diamond f$-source. The identity of the $\diamond f$-source does not have to be known (different runs can have different $\diamond f$-sources). Furthermore, it is not required that the destination processes of the $\diamond f$-source be correct: some (or even all) of them may crash.
- Similarly, for $n > 2f$, consensus can be solved if there is at least one correct $\diamond f$-source.
- For the special case that $f = 1$, a common assumption in practice, to implement Ω or solve consensus we need only one $\diamond$-timely link $p \rightarrow q$. Furthermore, there is no need to know which link it is, and p or q is allowed to crash.
- The above results are in some sense optimal: neither Ω nor consensus can be solved if processes are "only" $(f-1)$-sources. This impossibility result holds even if all processes are $(f-1)$-sources and all links are reliable.

We also study the *communication overhead* of implementations of Ω. The first algorithm that we give to implement Ω (for systems with up to f crashes and at least one correct $\diamond f$-source) is simple but not efficient: every process sends messages to all the other processes, forever. In other words, all the $O(n^2)$ links transport messages, forever. This is undesirable and can be avoided. Intuitively, after a process becomes the common leader,[2] it must periodically send messages forever (because if it crashes, processes must be able to notice this failure and chose a new leader); but from then on no other process needs to be monitored. Thus, after processes agree on a common leader, no process other than the leader should have to send messages. In fact, several papers have focused on reducing the communication overhead required to implement Ω in various systems [16, 1, 3], so that only $n-1$ links carry messages forever.

In this paper, we give an algorithm for Ω such that eventually only f links carry messages. This algorithm works for systems with reliable links where at most $f < n$ process may crash and there is at least one (unknown) correct $\diamond f$-source. A simple proof shows that in such systems, no implementation of Ω can ensure that fewer than f links carry messages forever, and so

[2] Note that processes may never know whether this has already occurred.

our algorithm is communication-optimal in this sense. It is worth noting that in the special case that $f = 1$, this algorithm for Ω requires only one unknown directed link to be timely (any one of its end-points may crash, and all the other links can be asynchronous), and eventually only one directed link carry messages (all the other links become quiescent).

Related work

Related work concerns solving consensus and implementing Ω in various types of partially synchronous systems, as well as the use of Ω to solve consensus. It is well-known that consensus cannot be solved in a completely asynchronous system (where both processes and links are asynchronous)[11]. Following this impossibility result, [9, 10] proposed several models of partial synchrony and for each one, they determined whether consensus was solvable. In particular, it was established that if all links are asynchronous then consensus remains unsolvable even if all processes are synchronous; on the other hand, if all links are eventually timely, then consensus can be solved.

Another approach to circumvent the impossibility result of [11] is the use of unreliable failure detectors [7]. In particular, Ω was shown to be the weakest failure detector that can be used to solve consensus in systems with a majority of correct processes [6], and it is the failure detector required by several algorithms [5, 12, 13, 14, 15, 17, 19]. For these reasons there is growing interest in implementing Ω efficiently in various types of partially synchronous system [16, 1, 3].

Several papers have focused on reducing the communication overhead of implementations of Ω. In particular, [16, 1, 3] give algorithms that require only $n - 1$ links to carry messages forever. These algorithms, which tolerate any number of process crashes, differ by the strength of the assumptions they make on the underlying systems: in [16] all links are $\diamond$-timely in both directions, in [1] all links to and from at least one correct process are $\diamond$-timely, and in [3] all links from at least one correct process are $\diamond$-timely. These algorithms also differ by the type of link failures, if any, they tolerate.

A different approach to the implementation of failure detectors was introduced in [20]. Instead of assuming (eventual) bounds on process and communication delays, it assumes that the message passing system has the following behavior: roughly speaking, there is a correct process p and a set Q of $f + 1$ processes such that if processes repeatedly wait to receive messages from $n - f$ processes (i.e., they proceed in "asynchronous rounds"), then eventually the messages from p are always among the first $n - f$ messages received by each process in Q. This assumption is incomparable to assuming the existence of an f-source (or a $\diamond f$-source): even if all processes are f-sources, every correct process may be excluded infinitely often from the sets of $n - f$ messages that processes in Q receive first (e.g., this would occur if all the links have bounded delays, but the currently slowest one rotates forever in a "round-robin" fashion among all the links).

[18, 21] defined families of failure detectors, denoted S_k and $\diamond S_k$, that have strong completeness and (eventual) limited accuracy; roughly speaking, parameter k is the number of processes for which accuracy applies. [18, 21] show how to use S_k and $\diamond S_k$ to solve k-set agreement problem, and [4] determines under which conditions (in terms of the maximum number of process crashes f) S_k and $\diamond S_k$ can be transformed to S and $\diamond S$ — two failure detectors whose accuracy applies to all processes [7]. This later work is similar but incomparable to the lower bounds in this paper: it concerns failure detector transformations in purely asynchronous systems, while the lower bounds derived here refer to various types of partially synchronous systems (e.g., our lower bounds refer to systems where *every* process is an $(f - 1)$-source — a property that cannot be expressed in terms of S_k or $\diamond S_k$ failure detectors).

Roadmap

In Section 2, we describe our model and define the leader failure detector Ω. In Section 3, we describe an algorithm for Ω for systems with up to f crashes where links are fair lossy, but some correct $\diamond f$-source exists; this result immediately implies that, when $n > 2f$, consensus can be solved in such systems [2]. In Section 4, we show that the above results are optimal in a precise sense: even if all processes are $(f-1)$-sources, neither Ω nor consensus can be solved. In Section 5, we give an algorithm for Ω (for systems where links are reliable) such that eventually only f links carry messages, and we show that this is optimal. Due to the space restrictions, proofs are omitted; they will be included in the full version of this paper.

2. INFORMAL MODEL

We consider distributed systems with $n \geq 2$ processes $\Pi = \{0, \ldots, n-1\}$ that can communicate with each other by sending messages through a set of di-

rected links Λ. To simplify the exposition of the model, we assume the existence of a discrete global clock which is not accessible to the processes. The range $\mathcal{T}$ of the global clock ticks is the set of natural numbers.

Processes. Processes execute by taking steps. In a step a process can either receive a set of messages and then change its state, *or* it can send a message and then change its state.[3]

A process can fail by permanently crashing, in which case it stops taking steps. A *process failure pattern* F is a function from $\mathcal{T}$ to 2^{Π}. Intuitively, $F(t)$ denotes the set of processes that have crashed through time t. Once a process crashes, it does not recover, i.e., $\forall t : F(t) \subseteq F(t+1)$. We say that *p crashes (or is faulty) in* F if $p \in F(t)$ for some t; otherwise we say that p *is correct in* F. A *process* p *is alive at time* t if $p \notin F(t)$.

We assume there is a lower and upper bound on the rate of execution (number of steps per time unit) of any non-faulty process. Processes have local clocks that are not necessarily synchronized, but we assume that they can accurately measure intervals of time (it is easy to extend our results to local clocks with bounded drift rates).

Links. We assume that the network is fully connected, i.e., $\Lambda = \{(p,q) \mid p,q \in \Pi \text{ and } p \neq q\}$. The directed link from process p to process q is denoted by $p \rightarrow q$. Every link $p \rightarrow q$ satisfies the following *integrity* property: q receives a message m from p at most once, and only if p previously sent m to q.[4]

In addition to integrity, links may have other properties concerning their degree of reliability and synchrony. We consider four types of links.

A link $p \rightarrow q$ is *reliable* if it satisfies integrity and also the following *no loss* property: If p sends a message m to q and q is correct, then q eventually receives m from p.

We also consider links that can intermittently drop messages, provided that the links satisfy a *fairness* property. To define this property, we assume that messages carry a *type* in addition to its *data*. Fairness requires that if a process sends an infinite number of messages of a type through a link then the link delivers an infinite number of messages of that type. More precisely, we assume that each message m consists of a pair $m = (type, data) \in \Sigma^* \times \Sigma^*$ where $\Sigma = \{0,1\}$. A link $p \rightarrow q$ is *fair lossy* if it satisfies integrity and also *fairness*: For every $type$, if p sends infinitely many messages of type $type$ to q and q is correct, then q receives infinitely many messages of type $type$ from p. Note that a reliable link is also fair lossy.

We consider two synchrony properties (the first is stronger than the second). A link $p \rightarrow q$ is *timely* if it satisfies integrity and also *timeliness*: There exists δ such that if p sends a message m to q at time t and q is correct, then q receives m from p by time $t + \delta$. The maximum message delay δ associated with a timely link is *not* known.

A link $p \rightarrow q$ is $\diamond$*-timely* if it satisfies integrity and also $\diamond$*-timeliness*: There exists δ and a time t such that if p sends a message m to q at time $t' \geq t$ and q is correct, then q receives m from p by time $t' + \delta$. The maximum message delay δ and the time t after which it holds *are not* known. Moreover, messages sent before time t can be lost.[5]

Note that a timely link is also reliable, and a $\diamond$-timely link is also fair lossy. Moreover, a link (that satisfies integrity) is trivially timely or $\diamond$-timely if one of its endpoint crashes. More precisely, link $p \rightarrow q$ is trivially timely if p is crashed at time 0 or q is faulty. Similarly, link $p \rightarrow q$ is trivially $\diamond$-timely if p crashes at time $t > 0$.

In summary, links can be reliable, fair lossy, timely or $\diamond$-timely. To indicate which links have which properties, we introduce a *link type function* L that maps each link in Λ to $\{R, FL, T, ET\}$: For link $\ell = p \rightarrow q$, $L(\ell) = R$, FL, T or ET implies that ℓ is reliable, fair lossy, timely or $\diamond$-timely, respectively.

Behaviors: A *behavior* $B = (F, L)$ combines a process failure pattern and a link type function: it indicates which processes (if any) fail, and at which time they do; and it also indicates which links are reliable, fair lossy, timely or $\diamond$-timely. We assume that L is consistent with F: a link $p \rightarrow q$ such that p and/or q crashes according to F is timely or $\diamond$-timely according to L, depending on the crash time(s) as explained above.

A system is characterized by a *set* of possible behaviors: these describe the process failures and the link types that can occur in a run of this system.

Sources. A process p is a j*-source [*$\diamond j$*-source] in a*

[3] Our lower bounds also hold in a stronger model in which a process can receive, change state, *and* send in a single atomic step.

[4] We assume that messages are unique, e.g., each message contains the id of the sender and a sequence number (this is implicit in all our algorithms).

[5] This $\diamond$-timeliness property corresponds to the $\mathcal{M}_3$ model of partial synchrony defined in [7].

behavior $B = (F, L)$ if p has j output links that are timely [$\diamond$-timely] according to L. We say that a *system has a j-source [$\diamond j$-source]* if every behavior B of that system has a j-source [$\diamond j$-source].

Note that j-sources and the destinations of j-sources are not necessarily correct processes. More precisely, suppose that in a given behavior $B = (F, L)$, p is a j-source, and $p_1, \ldots, p_j$ are the processes such that, for every $1 \leq k \leq j$, the link $p \rightarrow p_k$ is timely. It is still possible for p and/or some of the p_k's to be faulty in F. But when p is a correct process, we say that p is a *correct* j-source. The same remarks applies to any $\diamond j$-source and its corresponding destination processes.

The leader failure detector Ω. The formal definition of failure detector Ω is given in [7, 6]. Informally, at each process p, the failure detector module of Ω at p outputs the identity of a single process, denoted $leader_p$, such that the following property holds:

- There exists a correct process ℓ and a time after which, for every alive process p, $leader_p = \ell$.

If at time t, $leader_p$ contains the same correct process ℓ for all alive processes p, then we say that ℓ *is the leader at time* t. Note that at any given time processes do not know if there is a leader; they only know that eventually a leader emerges and remains.

3. IMPLEMENTING Ω AND CONSENSUS WITH A CORRECT $\diamond f$-SOURCE

In this section we consider systems where links are fair lossy, up to f processes may crash, and at least one correct process is an $\diamond f$-source (the identity of this $\diamond f$-source is unknown, and different runs may have different $\diamond f$-sources). We show that in such systems one can implement Ω and solve consensus (for consensus $n > 2f$ is also required). If $f = 1$, an important case in practice, this implies that the existence of a single unknown $\diamond$-timely link $p \rightarrow q$ is sufficient to implement Ω and solve consensus, even if process p or q crashes.

3.1 Implementing Ω

Figure 1 shows an algorithm for Ω for systems where links are fair lossy, up to f processes may crash and at least one correct process is an $\diamond f$-source. Each process p keeps a monotonically nondecreasing value *counter*$[q]$ for each process q. Process p chooses as its leader ℓ the process with smallest *counter*$[\ell]$, breaking ties using the process id. Each process periodically sends to other processes an ALIVE message with its *counter* vector. When a process receives such a message, it updates its *counter* vector by taking the component-wise $\max$. If a process p has not received ALIVE from q for *Timeout*$[q]$ time units, it increments *Timeout*$[q]$, in case the timeout was premature, and then sends $(\text{SUSPECT}, q)$ to all[6]. When a process p receives $(\text{SUSPECT}, q)$ from r, it adds r to the list of processes that have suspected q. If the size of such a list is at least $n - f$, p resets the list to $\emptyset$, and increments *counter*$[q]$.

THEOREM 1. *For all $n > f$, Ω can be implemented in systems with fair lossy links and n processes where up to f processes may crash and at least one (unknown) correct process is a $\diamond f$-source.*

Now consider a system where up to f processes may crash, no correct process is a $\diamond f$-source, but there exists a correct process p and f other distinct processes such that p is connected to the f processes through $\diamond$-timely links and correct processes. Then, by simple flooding of messages, we can simulate a system where p is actually a correct $\diamond f$-source, and then we can use the algorithm of Figure 1 to implement Ω.

COROLLARY 2. *For all $n > f$, Ω can be implemented in systems with fair lossy links and n processes where up to f processes may crash and at least one (unknown) correct process is connected to f other processes through a path of $\diamond$-timely links and correct processes.*

3.2 Solving consensus

It is well-known that Ω can be used to solve consensus if $n > 2f$ and links are reliable [7, 6] or fair lossy [2]. Together with Theorem 1, this implies that having at least one correct $\diamond f$-source is sufficient to solve consensus:

THEOREM 3. *For all $n > 2f$, consensus can be solved in systems with fair lossy links and n processes where up to f processes may crash and at least one (unknown) correct process is a $\diamond f$-source.*

By using message flooding we also get the following:

COROLLARY 4. *For all $n > 2f$, consensus can be solved in systems with fair lossy links and n processes*

[6]By convention, when a process sends a message to all, it sends the message to all processes except itself and it simulates sending to itself, that is, the process executes the code that handles the receipt of the message.

CODE FOR EACH PROCESS p:

on initialization:
1 $\forall q \neq p$: *Timeout*$[q] \leftarrow \eta + 1$
2 $\forall q$: *counter*$[q] \leftarrow 0$, *suspect*$[q] \leftarrow \emptyset$
3 $\forall q \neq p$: reset *timer*(q) to *Timeout*$[q]$
4 **start tasks** 0, 1 and 2

task 0:
5 **repeat forever**
6 *leader* $\leftarrow \ell$ such that $(\textit{counter}[\ell], \ell) = \min\{(\textit{counter}[q], q) : q \in \Pi\}$

task 1:
7 **repeat forever**
8 send $(\text{ALIVE}, \textit{counter})$ to all processes except p every η time

task 2:
9 **upon** receive (ALIVE, c) from q **do**
10 **for each** $r \in \Pi$ **do** *counter*$[r] \leftarrow \max\{\textit{counter}[r], c[r]\}$
11 reset *timer*(q) to *Timeout*$[q]$

12 **upon** expiration of *timer*(q) **do**
13 *Timeout*$[q] \leftarrow$ *Timeout*$[q] + 1$
14 send $(\text{SUSPECT}, q)$ to all
15 reset *timer*(q) to *Timeout*$[q]$

16 **upon** receive $(\text{SUSPECT}, q)$ from r **do**
17 *suspect*$[q] \leftarrow$ *suspect*$[q] \cup \{r\}$
18 **if** $|\textit{suspect}[q]| \geq n - f$ **then**
19 *suspect*$[q] \leftarrow \emptyset$
20 *counter*$[q] \leftarrow$ *counter*$[q] + 1$

Figure 1: Implementing Ω in systems with fair lossy links and at least one correct $\diamond f$-source.

where up to f processes may crash and at least one (unknown) correct process is connected to f other processes through a path of $\diamond$-timely links and correct processes.

3.3 The special case $f = 1$

Theorems 1 and 3 have a surprising corollary for systems where at most one process may crash — a common assumption in practice. For such systems, these theorems imply that the existence of a *single*, unknown $\diamond$-timely link $p \rightarrow q$ is sufficient for implementing Ω and solving consensus, *even if p or q crash.* To see this, suppose link $p \rightarrow q$ is $\diamond$-timely (in any given behavior). There are two possible cases. If p is correct (in that behavior), then p is a correct $\diamond 1$-source, and by Theorems 1 and 3, Ω can be implemented and consensus can be solved. If p is faulty, then, since $f = 1$, q must be correct (in this behavior). Consider the link $q \rightarrow p$. Since p eventually crashes, this link is trivially $\diamond$-timely. So, q is a correct $\diamond 1$-source, and again by Theorems 1 and 3, Ω can be implemented and consensus can be solved. Thus we have the following:

COROLLARY 5. *Consider any system with $n > 2$, $f = 1$, and fair lossy links. If there is (at least) one unknown directed link that is $\diamond$-timely, then Ω can be implemented and consensus can be solved. This holds even if the source or destination of that link crashes.*

4. $(f-1)$-SOURCES ARE NOT SUFFICIENT FOR IMPLEMENTING Ω OR SOLVING CONSENSUS

Consider systems where $n > f \geq 1$. In the previous section, we showed that having at least one correct $\diamond f$-source is sufficient to implement Ω and solve consensus (even if links are fair lossy). In this section, we state that this is a necessary condition, in the following sense: neither Ω nor consensus can be solved in a system where processes are "only" $(f-1)$-sources. This

holds even if we assume that all links are reliable, and *all* processes are $(f-1)$-sources.

To state these impossibility results succinctly, it is convenient to define system $S_{n,f,f-1}$. This is the system with n processes that has all the possible behaviors such that up to f processes may crash, every process is a $f-1$-source, and all the links are reliable. Thus, in every run of $S_{n,f,f-1}$, every process has at least $f-1$ outgoing links that are timely.

THEOREM 6. *For all $n > f \geq 1$, Ω cannot be implemented in system $S_{n,f,f-1}$.*

THEOREM 7. *For all $n > f \geq 1$, consensus cannot be solved in system $S_{n,f,f-1}$.*

5. COMMUNICATION-OPTIMAL IMPLEMENTATION OF Ω

The algorithm that implements Ω in Figure 1 is simple but inefficient: it requires all $n(n-1)$ links to carry message forever. In this section, we describe an algorithm for Ω such that eventually only f links carry messages. This algorithm, shown in Figure 2, works in systems with reliable links where up to f processes may crash and at least one correct process is a f-source. [7] This algorithm is communication-optimal in the sense that every implementation of Ω for such systems requires at least f links to carry messages forever in some runs (see Theorem 9).

Like in the previous algorithm of Figure 1, in the algorithm of Figure 2 each process p keeps a vector $counter_p[q]$ of integers, which roughly counts the number of times that q has been suspected by $n-f$ processes. However, unlike the previous algorithm, the responsibility to increment this counter is delegated to process q. More precisely, when a process r has seen $n-f$ SUSPICIONS for q, it sends an ACCUSATION to q, telling q to increment its own $counter_q[q]$.

Like in the previous algorithm, the $counter_p[q]$ vector is used to choose the leader. Unlike the previous algorithm, the leader that p chooses is not the process q with smallest $counter_p[q]$ among *all* processes in the system, but only among the processes in a set $Contenders_p$. This set initially has only p, and it may later change as p receives ALIVE and SUSPICION messages from other processes.

A process p keeps sending ALIVE messages to indicate that it is alive. Unlike the previous algorithm, these messages do not have the complete *counter* vector, but only the entry for p. Moreover, not all processes keep sending ALIVE forever: p only sends ALIVE if p considers itself to be the leader. Thus, p may voluntarily choose to stop sending ALIVE. When that happens, other processes will time out on p and will subsequently send an ACCUSATION to p. We would like p to increment its $counter_p[p]$ only when the messages from p are slow, rather than when p voluntarily stops sending ALIVE.[8] To do so, p keeps a current phase number $ph_p[p]$ for itself, and a process q keeps its current estimate of p's phase in $ph_q[p]$. The ALIVE messages from p includes not only p's counter, but also p's phase. When q receives such a message, it updates $ph_q[p]$. SUSPICION and ACCUSATION also carry both a phase number and counter number. If p receives an ACCUSATION with a phase and counter that matches p's own (called an *up-to-date* ACCUSATION), p accepts the message an increments its counter. Else p simply ignores it. In this way, when p voluntarily stops sending ALIVE messages, p can ignore consequent ACCUSATIONS by incrementing its phase.

When process p sends ALIVE, it does not send to all. Since there are only f timely links, p only sends to f processes. But since p does not know which of its links are timely, it has to guess. It does so by picking an arbitrary set of f processes and, if p later learns that it picked incorrectly, it switches to a different set. More precisely, let $Q[p,0], Q[p,1], \ldots, Q[p,m-1]$ be an enumeration of all subsets of f processes not containing p. The number of such subsets is $m = \binom{n-1}{f}$. Process p rotates over these subsets using its $counter_p[p]$ variable modulo m. When p increments its counter or its phase, it sends *one* ALIVE to *all* processes (doing so is necessary so that all processes are informed of the new counter and phase of p). Afterwards, p keeps sending ALIVE only to processes in $Q[p, counter_p[p] \bmod m]$.

THEOREM 8. *For all $n > f$, the algorithm in Figure 2 implements Ω in systems with reliable links and n processes where up to f processes may crash and at least one (unknown) correct process is an f-source.*

[7] Note that for links that do not lose messages, the timeliness and $\diamond$-timeliness properties are actually equivalent (this is because these properties do *not* assume that the bound on message delay is known). Thus, a reliable link is $\diamond$-timely if and only if it is timely. So in systems where all links are reliable, as we assume in this section, a process is a $\diamond f$-source if and only if it is an f-source. For brevity, we express our results in this section in terms of f-sources.

[8] It turns out that if we did not do so, the algorithm would fail.

CODE FOR EACH PROCESS p:

on initialization:

1 $\forall q \neq p$: *Timeout*$[q] \leftarrow \eta + 1$
2 $\forall q \neq p$: *timer*$(q) \leftarrow$ *off*
3 $\forall q$: *ph*$[q] \leftarrow 0$; *counter*$[q] \leftarrow 0$
4 *Contenders* $\leftarrow \{p\}$
5 *leader* $\leftarrow p$
6 *last* $\leftarrow (-1, -1)$
7 **start tasks** 0, 1 and 2

task 0**:**

8 **repeat forever**
9 *leader* $\leftarrow \ell$ such that $(\textit{counter}[\ell], \ell) = \min\{(\textit{counter}[q], q) : q \in \textit{Contenders}\}$

task 1**:**

10 **repeat forever**
11 **while** *leader* $= p$ **do**
12 **if** *last* $\neq$ (*ph*$[p]$, *counter*$[p]$) **then**
13 send $(\text{ALIVE}, p, \textit{ph}[p], \textit{counter}[p])$ to all processes except p
14 *last* $\leftarrow$ (*ph*$[p]$, *counter*$[p]$)
15 **else** send $(\text{ALIVE}, p, \textit{ph}[p], \textit{counter}[p])$ to the processes in $Q[p, \textit{counter}[p] \bmod \binom{n-1}{f}]$ every η time
16 *ph*$[p] \leftarrow$ *ph*$[p] + 1$
17 send $(\text{ALIVE}, p, \textit{ph}[p] - 1, \textit{counter}[p])$ to all processes except p
18 **while** *leader* $\neq p$ **do nop**

task 2**:**

19 **upon** receive (ALIVE, q, i, c) from r **do**
20 **if** it is the first receipt of (ALIVE, q, i, c) **then**
21 send (ALIVE, q, i, c) to all processes except p, q and r
22 **if** $(i, c) \geq (ph[q], counter[q])$ **then**
23 *ph*$[q] \leftarrow i$
24 *counter*$[q] \leftarrow c$
25 reset *timer*(q) to *Timeout*$[q]$
26 **if** did not receive $(\text{SUSPICION}, q, i, c)$ from $n - f$ processes **then**
27 *Contenders* $\leftarrow$ *Contenders* $\cup \{q\}$

28 **upon** expiration of *timer*(q) **do**
29 **if** did not previously send $(\text{SUSPICION}, q, \textit{ph}[q], counter[q])$ **then**
30 send $(\text{SUSPICION}, q, \textit{ph}[q], counter[q])$ to all processes except q
31 *Timeout*$[q] \leftarrow$ *Timeout*$[q] + 1$

32 **upon** receive $(\text{SUSPICION}, r, i, c)$ **do**
33 **if** received $(\text{SUSPICION}, r, i, c)$ from $n - f$ processes **and**
did not previously send $(\text{ACCUSATION}, r, i, c)$ **then**
34 send $(\text{ACCUSATION}, r, i, c)$ to r
35 *Contenders* $\leftarrow$ *Contenders* $- \{r\}$

36 **upon** receive $(\text{ACCUSATION}, p, i, c)$ **do**
37 **if** *ph*$[p] = i$ **and** $counter[p] = c$ **then**
38 *counter*$[p] \leftarrow$ *counter*$[p] + 1$

Figure 2: Communication-optimal implementation of Ω for systems with reliable links and at least one correct f-source.

Moreover, there is a time after which only f links carry messages (these are a subset of the links from the elected leader).

The above algorithm is communication-optimal in the following sense:

THEOREM 9. *For all $n > f \geq 1$, in a system with reliable links and n processes where up to f processes may crash and some unknown correct process is an f-source, any implementation of Ω requires at least f links to carry messages forever in every failure-free run.*

From Theorems 8 and 9, we have the following:

COROLLARY 10. *For all $n > f \geq 1$, the algorithm in Figure 2 is a communication-optimal implementation of Ω in systems with reliable links and n processes where up to f processes may crash and at least one (unknown) correct process is an f-source.*

6. A FINAL REMARK

In this paper, we showed how to implement Ω in systems with limited link synchrony (namely, systems with up to f crashes and at least one correct $\diamond f$-source), and we gave a communication-optimal implementation of Ω for such systems. We could have instead implemented Ω in these systems with the following three-step approach: first showing how to implement $\diamond S_f$, the limited accuracy failure detector introduced in [18, 21], then transforming $\diamond S_f$ into $\diamond S$ using the algorithm given in [4], and finally transforming $\diamond S$ into Ω using the algorithm given by Chu in [8]. We believe that the direct implementation of Ω given in Figure 1 is much simpler and clearer than the one obtained by the above approach. Moreover, the implementation of Ω that we give in Figure 2 is much more efficient: after the leader is elected only f links carry messages, while the implementation of Ω obtained by the three-step approach requires that $n(n-1)$ links send messages forever.

Acknowledgements

We are grateful to Michel Raynal for his comments, which helped improve this paper.

7. REFERENCES

[1] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Stable leader election (extended abstract). In *Proceedings of the 15th International Symposium on Distributed Computing*, LNCS 2180, pages 108–122. Springer-Verlag, 2001.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, Apr. 2000.

[3] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. On implementing omega with weak reliability and synchrony assumptions. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*, pages 306–314, Boston, Massachusetts, USA, July 2003.

[4] E. Anceaume, A. Fernandez, A. Mostefaoui, G. Neiger, and M. Raynal. A necessary and sufficient condition for transforming limited accuracy failure detectors. *Journal of Computer and System Sciences*, 2003(to appear).

[5] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, Nov. 2002.

[6] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[7] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, Mar. 1996.

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

[9] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, Jan. 1987.

[10] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[12] E. Gafni and L. Lamport. Disk paxos. In *Proceedings of the 14th International Symposium on Distributed Computing*, LNCS 1914, pages 330–344. Springer-Verlag, 2000.

[13] R. Guerraoui and P. Dutta. Fast indulgent consensus with zero degradation. In *Proceedings*

of the 4th European Dependable Computing Conference, Oct. 2002.

[14] L. Lamport. The Part-Time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[15] L. Lamport. Paxos made simple. *SIGACT News*, 32(4):18–25, Dec. 2001.

[16] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th Symposium on Reliable Distributed Systems*, pages 52–59. IEEE Computer Society Press, Oct. 2000.

[17] M. Larrea, A. Fernandez, and S. Arevalo. Eventually consistent failure detectors. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 326–327, 2001.

[18] A. Mostéfaoui and M. Raynal. k-set agreement with limited accuracy failure detectors. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 143–152, aug 2000.

[19] A. Mostéfaoui and M. Raynal. Leader-based consensus. *Parallel Processing Letters*, 11(1):95–107, 2001.

[20] E. Mourgaya, A. Mostefaoui, and M. Raynal. Asynchronous implementation of failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks*. IEEE Computer Society Press, 2003.

[21] J. Yang, G. Neiger, and E. Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing*, pages 297–306, 1998.