

# Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty

HAGIT ATTIYA

*Technion, Haifa, Israel*

CYNTHIA DWORK

*IBM Almaden Research Center, San Jose, California*

NANCY LYNCH

*Massachusetts Institute of Technology, Cambridge, Massachusetts*

AND

LARRY STOCKMEYER

*IBM Almaden Research Center, San Jose, California*

**Abstract.** Upper and lower bounds are proved for the time complexity of the problem of reaching agreement in a distributed network in the presence of process failures and inexact information about time. It is assumed that the amount of (real) time between any two consecutive steps of any nonfaulty process is at least  $c_1$  and at most  $c_2$ ; thus,  $C = c_2/c_1$  is a measure of the timing uncertainty. It is also assumed that the time for message delivery is at most  $d$ . Processes are assumed to fail by stopping, so that process failures can be detected by timeouts.

A straightforward adaptation of an  $(f + 1)$ -round round-based agreement algorithm takes time  $(f + 1)Cd$  if there are  $f$  potential faults, while a straightforward modification of the proof that  $f + 1$  rounds are required yields a lower bound of time  $(f + 1)d$ . The first result of this paper is an agreement algorithm in which the uncertainty factor  $C$  is only incurred for *one* round, yielding

---

A preliminary version of this work appeared in *Proceedings of the 23rd ACM Symposium on Theory of Computing* (New Orleans, La., May 6–8), ACM, New York, 1991, pp. 359–369.

This work was supported in part by Office of Naval Research (ONR) contracts N00014-85-K-0168 and N00014-91-J-1046, by National Science Foundation (NSF) grants CCR 86-11442 and CCR 89-15206, and by DARPA contracts N00014-87-K-0825, N00014-89-J-1988 and DARPA-N00014-92-J-4033.

The work of H. Attiya was done in part while at the Laboratory for Computer Science, MIT. The work of C. Dwork and L. Stockmeyer was done in part while on sabbatical at the Laboratory for Computer Science, MIT.

Authors' addresses: H. Attiya, Department of Computer Science, Technion, Haifa, Israel; C. Dwork, Department K53/802, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120; N. Lynch, Laboratory for Computer Science, MIT, Cambridge, MA 02139; L. Stockmeyer, Department K53/802, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1994 ACM 0004-5411/94/0100-0122 \$03.50

Journal of the Association for Computing Machinery, Vol. 41, No. 1, January 1994, pp. 122–152

a running time of approximately  $2fd + Cd$  in the worst case. (It is assumed that  $c_2 \ll d$ .) The second result shows that any agreement algorithm must take time at least  $(f - 1)d + Cd$  in the worst case.

The new agreement algorithm can also be applied in a model where processors are synchronous ( $C = 1$ ), and where message delay during a particular execution of the algorithm is bounded above by a quantity  $\delta$  which could be smaller than the worst-case upper bound  $d$ . The running time in this case is approximately  $(2f - 1)\delta + d$ .

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems—*distributed applications; distributed databases; network operating systems*; C.4 [Performance of Systems]: *reliability; availability; and serviceability*

General Terms: Algorithms, Performance, Reliability, Theory

Additional Key Words and Phrases: Agreement, consensus, distributed agreement, distributed consensus, fault-tolerance, timeout, timing uncertainty

## 1. Introduction

Distributed computing theory has studied the complexity requirements of many problems in synchronous and asynchronous models of computation. There is an important middle ground, however, between the synchronous and asynchronous extremes: models that include inexact information about timing of events. This middle ground is reasonable for modeling real distributed systems, in which the amount of time required for processes to take steps, for clocks to advance, and for messages to be delivered are generally only approximately known.

We are interested in determining the complexity of problems of the sort arising in distributed computing theory in models with inexact timing information. In particular, in this paper, we consider the time complexity of the problem of *fault-tolerant distributed agreement*. In the version of the agreement problem we consider, there is a system of  $n$  processes,  $p_1, \dots, p_n$ , where each  $p_i$  is given an input value  $v_i$ . Each process that does not fail must choose a decision value such that (i) no two processes decide differently and (ii) if any process decides  $v$  then  $v$  was the input value of some process. We assume that processes fail only by stopping. This abstract problem can be used to model a variety of problems in distributed computing, e.g., agreement on the value of a sensor in a real-time computing system or agreement on whether to commit or abort a transaction in a database system.

The time complexity of the distributed agreement problem has been well studied in the synchronous “rounds” model. In this model, the computation proceeds in a sequence of rounds of communication. In each round, each non-failed process sends out messages to all processes, receives all messages sent to it at that round, and carries out some local computation.<sup>1</sup> The most basic time-bound results in these papers are matching upper and lower bounds of  $f + 1$  on the number of synchronous rounds of communication required for reaching agreement in the presence of at most  $f$  faults.<sup>2</sup>

<sup>1</sup>See, for example, Berman et al. [1989], Coan [1986, 1987], DeMillo et al. [1982], Dolev et al. [1986], Dolev and Strong [1983], Dwork and Moses [1990], Fischer and Lynch [1982], Hadzilacos [1984], Lamport and Fischer [1982], Lamport et al. [1982], Merritt [1985], Moses and Tuttle [1988], Moses and Waarts [1988], and Pease et al. [1980] for results involving time complexity in this model.

<sup>2</sup>We use  $f$  to denote the number of faults, instead of the more traditional  $t$ , since we want to reserve  $t$  to denote time.

We consider how these bounds are affected by using, instead of the rounds model, one in which there is inexact timing information. In particular, we assume that the amount of time between any two consecutive steps of any nonfaulty process is at least  $c_1$  and at most  $c_2$ , where  $c_1$  and  $c_2$  are known constants; thus,  $C = c_2/c_1$  is a measure of the timing uncertainty. We also assume that the time for message delivery is at most  $d$ .<sup>3</sup> Since processes are assumed to fail only by stopping, process failures can be detected by “timeouts”; that is, if an expected message from some process is not received within a sufficiently long time, then that process is known to have failed. The time required to implement a timeout is roughly  $Cd$ . (We sometimes approximate time bounds under the assumption that  $c_2 \ll d$ . The formal statements of our results give the exact bounds.)

Initially, we hoped to be able to adapt known results about the rounds model to obtain good bounds for the version with inexact timing. Indeed, an  $(f + 1)$ -round algorithm can be adapted in a straightforward way to yield an algorithm for the timing-based model that requires time at most  $(f + 1)Cd$  if there are  $f$  potential faults. On the other hand, a simple modification of the proof that  $f + 1$  rounds are necessary yields a lower bound of time  $(f + 1)d$ . There is a significant gap between these two bounds in case  $C > 1$ , namely, a multiplicative factor equal to the timing uncertainty,  $C$ . The motivation for our work is to obtain closer bounds on the time complexity of this problem, in particular, to understand how this complexity depends on  $C$ .

The first result of this paper is an agreement algorithm in which the uncertainty factor  $C$  is only incurred for *one* round, yielding a running time of approximately  $2fd + Cd$  in the worst case. This algorithm uses timing information in a novel way in order to achieve fast time performance. An interesting feature of the algorithm is that it can be viewed as an asynchronous algorithm that uses a fault detection (specifically, a timeout) mechanism. That is, the timing bounds,  $c_1$ ,  $c_2$  and  $d$  are used only in the fault detection mechanism.

The second result shows that any agreement algorithm must take time at least  $(f - 1)d + Cd$  in the worst case. The proof of this lower bound combines ideas used in the rounds model [Coan and Dwork, 1991; DeMillo et al., 1982; Dolev and Strong, 1983; Dwork and Moses, 1990; Fischer and Lynch, 1982; Hadzilacos, 1984; Lamport and Fischer, 1982; Merritt, 1985] in the asynchronous model [Dolev et al., 1987; Fischer et al., 1985], and in timing-based models [Attiya and Lynch, 1989]. More specifically, it uses a “chain argument” such as those used previously to prove that  $f + 1$  rounds are required in the synchronous model, a “bivalence argument” such as those used previously to prove that fault-tolerant agreement is impossible in an asynchronous system, and a “time stretching” argument such as those used to prove lower bounds for resource allocation problems.

Although these bounds are not completely tight, they do demonstrate that the time complexity only involves the “timeout bound”  $Cd$  in a single additive term;  $Cd$  is not multiplied by  $f$  (the total number of potential failures) as it is in the naive algorithm. Note that this new bound represents a significant improvement over the naive algorithm in case  $C$  is large (greater than 2), as

<sup>3</sup>Results of Fischer et al. [1985] and Dolev et al. [1987] imply that if either one of the bounds  $c_1$  or  $d$  does not exist, then there is no agreement algorithm tolerant to even one fault. In the case that only  $c_2$  does not exist, agreement tolerant to one fault is impossible assuming that receiving and sending are not part of the same atomic step.

might happen in the presence of inaccurate processor clocks or variable-time process swapping.

Although our initial motivation was to understand the time to reach agreement in models where there is uncertainty in process step time, our new algorithm can be applied to more general situations. As noted above, our algorithm works in an asynchronous system with a fault detection (timeout) mechanism. The running time of the algorithm is expressed in terms of a *timeout bound*  $T$ , an upper bound on the elapsed time between the failure of a process and the time at which all correct processes detect the failure. Again, a straightforward modification of a rounds-based algorithm gives an upper bound of time  $(f + 1)T$ . In the new algorithm, the timeout bound  $T$  enters as a single additive term (not multiplied by  $f$ ).

Another application of our algorithm yields upper bound results for a related model used by Herzberg and Kuttan [1989] to study fault detection in host-to-host protocols. In their model, process steps are completely synchronous, that is,  $C = 1$ , and there is, as above, an upper bound  $d$  on the worst-case time for any message to be delivered. Even though algorithms must be designed to be correct in the case that any message delay is  $d$ , in reality, message delivery could be much faster than  $d$  in many executions. Therefore, it makes sense to express the time complexity of an algorithm in terms of a new parameter  $\delta$ , the actual message delay during execution of the algorithm, as well as in terms of the worst-case bound  $d$ . Again, a straightforward adaptation of an  $(f + 1)$ -round agreement algorithm gives an agreement algorithm for this model that runs in time  $(f + 1)d$ , even in executions where  $\delta \ll d$ . In contrast, the main agreement algorithm of this paper runs in time approximately  $(2f - 1)\delta + d$ . That is, the number of faults multiplies the actual message delay  $\delta$  rather than the worst-case delay  $d$ . Our lower bound techniques can be modified to give a lower bound of time  $(2f - n)\delta + d$ , if  $n \leq 2f$ , for this model [Dwork and Stockmeyer, 1991].

There has, of course, been a considerable amount of previous work on the agreement problem in various models; a representative selection of references to this work appears above. However, there has been very little work so far on this problem with inexact timing information.

Some prior work on distributed agreement in a model with inexact timing information appears in [Dwork et al., 1988]. The main emphasis in [Dwork et al., 1988] was on determining the maximum fault tolerance possible for various fault models; only rough upper bounds on the time complexity of the algorithms were given, and no lower bounds on time were proved. In contrast, the main emphasis of the present paper is on time complexity.

Related work on the *latency* of reaching agreement when processes are not completely synchronous appears in [Cristian et al., 1985] and [Strong et al., 1990]. (The latency is defined to be the worst-case elapsed time as measured on the clock of any correct process.) These papers assume that process clocks are synchronized to within some fixed additive error, and the case  $\delta < d$  is not considered. Unlike the results in our paper, these results are stated in terms of clock time rather than absolute real time. Although it is possible to translate results from those papers into our model, doing so appears to yield results with a less precise dependency on the timing uncertainty than we obtain here. (More detailed comparison between the models was performed by Ponzio and Strong [1992].)

This work is part of an emerging study of the real-time behavior of distributed systems. Other work in this area includes the extensive literature on clock synchronization algorithms.<sup>4</sup> More recently, the mutual exclusion problem has been studied in a timing-based model with  $C > 1$  [Attiya and Lynch, 1989]. Also, the time complexity for a synchronizer algorithm to operate in a timing-based network is studied in [Attiya and Mavronicolas: 1990] and [Rhee and Welch, 1992], and the time complexity of leader election algorithms in a timing-based model appears in [Coan and Thomas, 1990].

The rest of the paper is organized as follows: Section 2 contains a description of the formal model we use for timing-based distributed systems and a statement of the distributed agreement problem. In Section 3, we describe a useful “subroutine” for timing out failed processes. Section 4 contains a discussion of some simple upper-bound results that arise easily from the known results for the rounds model. In Section 5, we give our main upper-bound result. Section 6 contains our lower-bound result. Section 7 contains our results for the model with synchronous processes and uncertain message delivery time. Finally, Section 8 contains our conclusions.

## 2. Definitions

**2.1. FORMAL MODEL.** In this section, we present the definitions for the underlying formal model.<sup>5</sup>

An *algorithm* consists of  $n$  processes  $p_1, \dots, p_n$ . Each process  $p_i$  is modeled as a (possibly infinite) state machine with state set  $Q_i$ . The state set  $Q_i$  contains a distinguished *initial state*  $q_{0,i}$  and a distinguished *fail state*.

A *configuration* is a vector  $C = (q_1, \dots, q_n)$  where  $q_i$  is the local state of  $p_i$ ; denote  $state_i(C) = q_i$ . The *initial configuration* is the vector  $(q_{0,1}, \dots, q_{0,n})$ . Processes communicate by sending messages (taken from some alphabet  $\mathcal{M}$ ) to each other. A *send action*  $send(j, m)$  represents the sending of message  $m$  to  $p_j$ . Let  $\mathcal{S}$  denote the set of all send actions  $send(j, m)$  for all  $m \in \mathcal{M}$  and all  $1 \leq j \leq n$ . Processes can receive *inputs* from some set  $\mathcal{V}$  of *values*.

We model a computation of the algorithm as a sequence of configurations alternated with *events*. Each event is either a *computation event*, representing a computation step of a single process, a *failure event*, representing the failure of some process, a *delivery event*, representing the delivery of a message to a process, or an input event, representing the arrival of a value at a process.

A *computation event* is specified by  $comp(i, S)$  where  $i$  is the index of the process taking the step and  $S$  is a finite subset of  $\mathcal{S}$ . In the computation step associated with event  $comp(i, S)$ , the process  $p_i$ , based on its local state, performs the send actions in  $S$  and possibly changes its local state. In all our algorithms the set of send actions will be *broadcast*( $m$ ), that is,  $\{send(1, m), \dots, send(n, m)\}$ . A broadcast includes a message to the sender itself. A *failure event* has the form  $fail(i, S)$  and causes the *send* actions in  $S$  to be performed; other properties of failure events are detailed below. Each delivery event has the form  $del(i, m)$  for some  $m \in \mathcal{M}$ , and each input event

<sup>4</sup>See Dolev et al. [1986]; Halpern et al. [1985]; Lamport and Melliar-Smith [1985]; Lundelius and Lynch [1984]; Srikanth and Toueg [1987]; Welch and Lynch [1988], for example.

<sup>5</sup>These definitions could be expressed in terms of the general *timed automaton model* described in [Attiya and Lynch, 1989] and [Merritt et al., 1991]; however, we choose here to present the definitions directly, in order to avoid the intervening layer of definitions.



has the form  $input(i, v)$  for some  $v \in \mathcal{V}$ . In these events, the process  $p_i$ , based on  $m$  (or  $v$ ) and its local state, may change its state.

Each process  $p_i$  follows a deterministic protocol that determines its state transitions and the messages it sends. In more detail, the protocol consists of two transition functions,  $\varphi_i$  for delivery and input events, and  $\gamma_i$  for computation events. For each  $q \in Q_i$  and  $a \in \mathcal{M} \cup \mathcal{V}$ ,  $\varphi_i(q, a)$  gives a state  $q' \in Q_i$ . For each  $q \in Q_i$ ,  $\gamma_i(q)$  gives a state  $q'$  and a finite set  $S$  of send actions. We assume in both cases that  $q = fail$  if and only if  $q' = fail$ , and we assume that  $S$  is empty if  $q = fail$ . These conditions mean intuitively that (i) the protocol cannot cause the process to leave the *fail* state, (ii) the protocol cannot cause a process to enter the *fail* state from a non-*fail* state, and (iii) no messages are sent from the *fail* state.

An *execution* is an infinite sequence of alternating configurations and events

$$\alpha = C_0, \pi_1, C_1, \dots, \pi_j, C_j, \dots,$$

satisfying the following conditions:

- (1)  $C_0$  is the initial configuration;
- (2) If  $\pi_j = del(i, a)$  or  $input(i, a)$ , then  $state_i(C_j)$  is obtained by applying  $\varphi_i$  to  $state_i(C_{j-1})$  and  $a$ ;
- (3) If  $\pi_j = comp(i, S)$ , then  $state_i(C_j)$  and  $S$  are obtained by applying  $\gamma_i$  to  $state_i(C_{j-1})$ ;
- (4) If  $\pi_j = fail(i, S)$ , then  $state_i(C_{j-1}) \neq fail$ ,  $state_i(C_j) = fail$ , and  $S$  is a subset of the send events obtained by applying  $\gamma_i$  to  $state_i(C_{j-1})$ ;
- (5) If  $\pi_j$  involves process  $i$ , then  $state_k(C_{j-1}) = state_k(C_j)$  for every  $k \neq i$ ;
- (6) (Each send is matched to a later delivery and each delivery to an earlier send.) For each  $m \in \mathcal{M}$  and each process  $p_i$ , let  $S(i, m)$  be the set of  $j$  such that  $\pi_j$  contains a  $send(i, m)$  and let  $D(i, m)$  be the set of  $j$  such that  $\pi_j$  is a delivery event  $del(i, m)$ . Then there is a bijective mapping  $\sigma_{i,m}$  from  $S(i, m)$  to  $D(i, m)$  such that  $\sigma_{i,m}(j) > j$  for all  $j \in S(i, m)$ .

A *timed event* is a pair  $(\pi, t)$ , where  $\pi$  is an event and  $t$ , the “time”, is a nonnegative real number. A *timed sequence* is an infinite sequence of alternating configurations and timed events

$$\alpha = C_0, (\pi_1, t_1), C_1, \dots, (\pi_j, t_j), C_j, \dots,$$

where the times are nondecreasing and unbounded.

Fix real numbers  $c_1$ ,  $c_2$ , and  $d$ , where  $0 < c_1 \leq c_2 < \infty$  and  $0 < d < \infty$ . Letting  $\alpha$  be a timed sequence as above, we say that  $\alpha$  is a *timed execution* provided that the following all hold:

- (1)  $C_0, \pi_1, C_1, \dots, \pi_j, C_j, \dots$  is an execution;
- (2) There are computation or failure events for all processes with time 0;
- (3) There are infinitely many computation or failure events for each process;
- (4) (Bounds on step time) Suppose  $j < k$ , the  $j$ th and  $k$ th timed events are both either computation or failure events of the same process  $p_i$ , and there are no intervening computation or failure events of  $p_i$ . Then  $c_1 \leq t_k - t_j \leq c_2$ ;
- (5) (Upper bound on message delivery time) If message  $m$  is sent to  $p_i$  at the  $j$ th timed event, then there exists  $k > j$  such that the  $k$ th timed event is the matching delivery  $(del(i, m), t_k)$  (i.e.,  $\sigma_{i,m}(j) = k$ ) and  $t_k - t_j \leq d$ .

Note that for any timed execution  $\alpha$  and any  $p_i$ , there is at most one timed event of the form  $(fail(i, S), t)$ . If there is such an event, we call  $t$  the *failure time* of  $p_i$ .

We define a *timed execution prefix* to be any finite prefix of a timed execution (ending with a configuration). For any timed execution prefix  $\alpha$ , we define  $t_{\text{end}}(\alpha)$  to be the time associated with the last event in  $\alpha$  (0 if  $\alpha$  contains no timed events).

We say that a process  $p_i$  *receives the message  $m$  by time  $t$*  (in a timed execution  $\alpha$ ) if, by time  $t$ ,  $p_i$  has a computation or failure event that is preceded in  $\alpha$  by a delivery event  $del(i, m)$ . For the rest of the paper let  $D$  denote  $d + c_2$ . Note that if  $m$  is sent to  $p_i$  at time  $t$ , then  $p_i$  receives  $m$  by time  $t + D$ . Similarly, we say that a process  $p_i$  *receives the input  $v$  by time  $t$*  if, by time  $t$ ,  $p_i$  has a computation or failure event that is preceded in  $\alpha$  by an input event  $input(i, v)$ .

For any timed execution  $\alpha$ , we define  $delay(\alpha)$  to be the maximum delay of any message delivery in  $\alpha$ . When  $\alpha$  is clear from context, we often use the notation  $\delta$  to denote  $delay(\alpha)$ , and will let  $\Delta = \delta + c_2$ .

To simplify the expression of our time bounds in terms of the parameters  $\delta$ ,  $d$ ,  $c_1$ , and  $c_2$ , we sometimes approximate the bounds in the case that  $c_2 \ll \delta$ . For example, in this case we have  $D \approx d$  and  $\Delta \approx \delta$ .

**2.2. THE AGREEMENT PROBLEM.** We now specify the *agreement problem*. The original definition of the problem in round-based systems (e.g., [Lamport et al., 1982]) assumes that all processes begin executing simultaneously with their initial values already in their states. This degree of initial synchronization is not very realistic in a distributed network. Since we are interested in capturing timing uncertainty, we have included *input* events in the definitions to permit asynchronous starts of the protocol. Let  $\mathcal{V}$  be a set of values. We assume that each set  $Q_i$  of local states includes a subset of *decision states* for each  $v \in \mathcal{V}$ , such that *fail* is not a decision state, the sets of decision states for different values are disjoint, and the transition functions  $\phi_i$  and  $\gamma_i$  map each decision state for  $v$  to a decision state for  $v$ . A process *decides on  $v$*  by changing its state to a decision state for  $v$  (so its state thereafter is always a decision state for  $v$ ).

A timed execution  $\alpha$  (or timed execution prefix) is *f-admissible* if  $\alpha$  contains at most  $f$  failure events and, for each  $p_i$ , exactly one input event  $input(i, v_i)$ . For each  $p_i$ , define  $start_i(\alpha)$  to be the smallest time  $t$  such that  $p_i$  receives an input by time  $t$ . Define  $start(\alpha)$  to be the maximum of  $start_i(\alpha)$  over all  $i$ . (It follows from the definition of receiving an input by time  $t$  that every process has had a computation or a failure event by  $start(\alpha)$ .)

Let  $B$  be a mapping from the positive reals to the positive reals. An algorithm *solves the agreement problem for  $f$  faults within time  $B$* , provided that each of its *f*-admissible timed executions  $\alpha$  satisfies the following:

- (1) (Agreement) No two different processes decide on different values;
- (2) (Validity) If some process decides on  $v$ , then an event  $input(i, v)$  occurs in  $\alpha$ ;<sup>6</sup>
- (3) (Termination and Time Bound) Every process either has a failure event or makes a decision by time  $start(\alpha) + B(delay(\alpha))$ .

<sup>6</sup>Note that this condition is slightly stronger than the usual validity condition for distributed agreement problems.

We finish this definition section with a statement of a slightly weaker version of the agreement problem. This may be interesting because our lower bound results still apply for the weaker problem statement. (Our upper bound, however, satisfies the stronger problem statement given above.) Namely, we define the *agreement problem with synchronized start* to be the same as the agreement problem, except that the three properties listed above need only hold for  $f$ -admissible timed executions  $\alpha$  in which each process receives its initial value at time 0; formally, for each process  $p_i$ , there is a timed event  $(input(i, v_i), 0)$  in  $\alpha$  that precedes every computation and failure event of  $p_i$ . Our default convention is that the synchronized start condition does *not* hold.

We carry out the main development using a Boolean version of the problem, that is,  $\mathcal{V} = \{0, 1\}$ . Later we discuss extensions to the case of an arbitrary value set.

### 3. A Timeout Strategy

In the algorithms, we describe below, it will be convenient to describe each  $p_i$  as a “parallel composition” of two tasks, a “timeout” task, and a “main” task.

The basic idea of the timeout task is very simple. At each step, each process broadcasts an *alive* message. If some process  $p_i$  has run for sufficiently many steps without receiving an *alive* message from the process  $p_j$ , then  $p_i$  concludes that  $p_j$  halted.

In more detail, the timeout task of  $p_i$  has the following state components: *blocked*, a Boolean, initially *true* (the purpose of *blocked* is to allow the main task to stop the timeout task); a set *halted*  $\subseteq \{1, \dots, n\}$ , initially  $\emptyset$ ; for each  $j \in \{1, \dots, n\}$  a nonnegative integer *counter*( $j$ ), initially  $-1$ . In addition, the local state of each process contains a component *buff*, to which messages are added at each message delivery event. Figure 1 describes the steps of the timeout task of process  $p_i$  that are associated with  $comp(i, S)$  events, in precondition-effect style. Recall that  $D = d + c_2$ .

Assume that each local protocol includes the transitions indicated in Figure 1. Say that a process *halts at time*  $t$  if it either fails at time  $t$  or sets *blocked* to *true* at time  $t$ . We assume that, if the main task of  $p_i$  sets *blocked* to *true* at some step, then the main task of  $p_i$  sends no messages at later steps. Fix a timed execution  $\alpha$ ; we prove the following properties for  $\alpha$ .

- T1. If any  $p_i$  adds  $j$  to *halted* at time  $t$ , then  $p_j$  halts, and every message sent from  $p_j$  to  $p_i$  is delivered strictly before time  $t$ .
- T2. There is a constant  $T$  such that, if  $p_j$  halts at time  $t$ , then every  $p_i$  either halts or adds  $j$  to *halted* by time  $t + T$ .

To verify T1, let  $p_i$  add  $j$  to *halted* at time  $t$ . We first show that  $p_j$  halts. If not, then  $p_j$  sends an *alive* message to  $p_i$  at each of its steps. The maximum difference between the times of two such consecutive send events is  $c_2$ ; the time between the two corresponding delivery events is maximized by assuming that the first message takes time 0 and the second takes time  $d$ . Thus, this difference is at most  $D$ . However, since time at least  $c_1$  elapses between every two steps of  $p_i$ , time at least  $c_1(\lfloor D/c_1 \rfloor + 1) > D$  must elapse between the last delivery of an *alive* message from  $p_j$  before time  $t$  and time  $t$  (when  $j$  is added to *halted*). This is a contradiction, so  $p_j$  halts.

By a similar argument, we show that every message from  $p_j$  to  $p_i$  gets delivered strictly before time  $t$ . Suppose that  $p_j$  sends a message  $m$  to  $p_i$  at some step. Then, at  $p_j$ 's previous step,  $p_j$  sends an *alive* message  $m'$  to  $p_i$ . As



**Precondition:***not blocked***Effect:****broadcast**((*alive*, *i*))**for** *j* := 1 **to** *n* **do***counter*(*j*) := *counter*(*j*) + 1**if** (*alive*, *j*) ∈ *buff* **then***remove* (*alive*, *j*) **from** *buff**counter*(*j*) := 0**elseif** *counter*(*j*) ≥ ⌊*D*/*c*<sub>1</sub>⌋ + 1 **then***add* *j* **to** *halted***od**

FIG. 1. The timeout task.

before, the maximum possible difference between the times of the deliveries of  $m'$  and of  $m$  is at most  $D$ , but time strictly greater than  $D$  must elapse between the delivery of  $m'$  and time  $t$ . It follows that  $m$  is delivered strictly before time  $t$ .

Now let  $\delta = \text{delay}(\alpha)$ , the maximum delay of any message delivery in  $\alpha$ , and recall that  $\Delta = \delta + c_2$ . We verify T2, with a timeout bound  $T$  of approximately  $Cd + \delta$ . Suppose  $p_j$  halts at time  $t$ , so that the last *alive* message from  $p_j$  to  $p_i$  is sent no later than time  $t$ . Therefore, by time  $t' = t + \Delta$ ,  $p_i$  will set  $p_j$ 's counter to zero for the final time. So by time  $t' + c_2(\lfloor D/c_1 \rfloor + 1)$ ,  $p_i$  adds  $j$  to *halted*. Therefore, our algorithm has the timeout bound

$$T = \Delta + c_2 \left( \left\lfloor \frac{D}{c_1} \right\rfloor + 1 \right).$$

In case  $c_2 \ll \delta$ , we have  $T \approx Cd + \delta$ .

In our algorithms that use the timeout task, we use only the fact that the timeout task has properties T1 and T2, and we express the time bounds of these algorithms in terms of the parameter  $T$ . Therefore, given a way to detect process failures with a timeout bound  $T$  smaller than the one given above, this detection method could be used to improve the time bounds. We do assume, however, that  $T \geq \Delta$ .

A technical point must be made concerning the parallel composition of the timeout task with the main task. Whenever a process takes a step, we imagine that a step of the timeout task is performed first, possibly adding new processes to *halted*. Then a step of the main task is performed, using the (possibly) new set *halted*. Even though this appears to be two transitions taken in sequence, it is easy to see that they can be combined into a single transition.

#### 4. Simple Bounds

In this section, we briefly discuss some simple algorithms for the agreement problem in the timing-based model and mention a simple lower bound.

We first give a method for transforming a round-based algorithm to an algorithm that works in the timing-based model.

Let  $A$  be a round-based algorithm involving processes  $p_i$  for  $1 \leq i \leq n$ . For each round  $r \geq 1$ , the local protocol of  $p_i$  determines the messages that  $p_i$  should send at round  $r$ , based on the messages received by  $p_i$  at rounds less than  $r$ . Assume that  $A$  runs for exactly  $R$  rounds and that every nonfaulty process sends a message to every process at every round 1 through  $R$ . (The transformation can be easily modified to allow some processes to halt earlier than the maximum round  $R$ .)

We describe an algorithm  $A'$  for the timing-based model. In this algorithm, each process includes a timeout task, as described in the previous section. Initially, each process sends its round-1 messages. Each  $p_i$  then waits, for each  $p_j$ , until it either receives the round-1 message of  $p_j$  or adds  $j$  to its set *halted*. Then  $p_i$  uses  $A$  to compute its round-2 messages, and these messages are sent. Subsequent rounds are handled similarly.

By Properties T1 and T2 of the timeout task, it should be clear that  $A'$  simulates  $A$  correctly. To bound the time of  $A'$ , let  $\alpha$  be an arbitrary  $f$ -admissible timed execution, and define real numbers  $t_r$  for  $0 \leq r \leq R$  as follows. (Each  $t_r$  will be shown to be an upper bound on the time for all nonhalted processes to complete the simulation of round  $r$ .) First,  $t_0 = \text{start}(\alpha)$ . Second, define  $t_1 = t_0 + T$  if some process has a failure event at some time  $t \leq t_0$ ; otherwise, define  $t_1 = t_0 + \Delta$ . Finally, for  $2 \leq r \leq R$ , define  $t_r = t_{r-1} + T$  if some process has a failure event at a time  $t$  with  $t_{r-2} < t \leq t_{r-1}$ ; otherwise, define  $t_r = t_{r-1} + \Delta$ . Since we assume  $T \geq \Delta$ , we have  $t_r \geq t_{r-1} + \Delta$  for all  $r \geq 1$ . It is also easy to see that, for every  $r$  such that a failure occurs at some time  $t \leq t_{r-1}$ ,  $t_r \geq u_{r-1} + T$  where  $u_{r-1}$  is the maximum time  $t \leq t_{r-1}$  such that a failure occurs at time  $t$ . By Property T2 of the timeout task, it follows easily by induction on  $r$  that every process either fails or completes round  $r$  no later than time  $t_r$  in the simulation of  $A$  by  $A'$ . If there are at most  $f$  faults, there are at most  $f$  values of  $r$  such that  $t_r = t_{r-1} + T$ . Therefore,  $A'$  takes time at most

$$T \cdot \min\{f, R\} + \Delta \cdot \max\{R - f, 0\}.$$

Taking  $A$  to be an  $(f + 1)$ -round agreement algorithm (such as the algorithm of Dolev and Strong [1983] appropriately modified for fail-stop faults), this transformation gives an upper bound of  $fT + \Delta$  on the time to solve the agreement problem with  $f$  faults. In the case that  $c_2 \ll \delta$ , this bound is approximately  $fCd + (f + 1)\delta$ .

In the case of synchronized start, there is another approach that does not perform the timeout task at every round but runs a related timing task to ensure that the entire algorithm runs long enough. The main agreement task in this case uses a “flooding” strategy. If a process  $p_i$  receives a 1 (at either an input event or a delivery event) and if  $p_i$  has not yet decided,  $p_i$  broadcasts the message 1 and decides 1. It is easy to see that, in any timed execution, if any correct process receives a 1, then some correct process receives a 1 no later than time  $fD$ . Since this correct process broadcasts a 1, all correct processes receive a 1 no later than time  $(f + 1)D$ . Therefore, any process that has run for time strictly more than  $(f + 1)D$  can decide 0. To ensure that this much time has elapsed, each process counts  $k = \lfloor (f + 1)D/c_1 \rfloor + 1$  of its own steps. This agreement algorithm takes time at most  $c_2k$ . This upper bound is approximately  $(f + 1)Cd$ . (This bound is better than the one for the simple simulation above when  $Cd < (f + 1)\delta$ .)

Note that both upper bounds contain the term  $fCd$ . Intuitively, this means that these algorithms can use  $f$  sequential “long” timeouts, where a long timeout takes time at least  $Cd$ . In the next section, we give a more subtle algorithm with a time bound that involves only one long timeout.

A lower bound of  $(f + 1)d$  is obtained fairly easily from the standard  $(f + 1)$ -round lower-bound proof in the round-based model [Coan and Dwork, 1991; Merritt, 1985]. The idea is to focus on executions in which processes take steps at every time  $c_2, 2c_2, \dots$ , and for every  $k$ , messages sent in the interval  $[(k - 1)d, kd)$  are delivered at time  $kd$ . If, for the sake of contradiction, we assume that some algorithm requires less than time  $(f + 1)d$ , then since no messages are delivered after time  $fd$  the processes must decide based on their states at time  $fd$ . This corresponds to deciding after  $f$  rounds of communication in the round-based model. The rest of the proof requires a small extension of the original proof to handle the fact that processes are taking multiple steps between successive deliveries. (Since the original proof requires  $f \leq n - 2$ , our lower bound has the same restriction.)

### 5. The Upper Bound

Now we present our main result, which shows how the upper bound can be improved so that  $Cd$  is not multiplied by  $f$  but only by 1.

**THEOREM 5.1.** *There is an algorithm to solve the agreement problem for  $f$  faults within time  $(2f - 1)\Delta + \max\{T, 3\Delta\}$ .*

Substituting the value of  $T$  obtained in Section 3, the following corollary is immediate.

**COROLLARY 5.2.** *There is an algorithm to solve the agreement problem for  $f$  faults within time  $2f\Delta + \max\{CD + c_2, 2\Delta\}$ .*

Assuming that  $c_2 \ll \delta$  and  $Cd \geq 2\delta$ , this upper bound is approximately  $2f\delta + Cd$ . If  $\delta = d$ , the bound is approximately  $2fd + Cd$ .

**5.1. THE ALGORITHM.** In addition to the local state components of the timeout process and *halted* and *blocked* (as described in Section 3), we assume that the local state of  $p_i$  contains components  $v_i$  and  $r$ , plus a component *buff* to hold incoming messages, plus a component to record decisions. The component  $v_i$  is the “input value component”, initially  $\perp$ ; an input event  $input(i, v)$  sets  $v_i$  to  $v$ . As in the timeout task, incoming messages are added to *buff* at each message delivery event. The component  $r$  holds a nonnegative integer *phase number*, initially 0. A **decide**( $v$ ) operation causes  $p_i$  to enter a decision state for value  $v$  (by recording the decision in the appropriate state component) and set *blocked* to *true* (to stop all nontrivial transitions, including those of the timeout task).

Now we give an informal description of the algorithm, or, more specifically, of the steps of process  $p_i$  that are associated with  $comp(i, S)$  events. The algorithm is given in more detail in Figure 2. This description and the associated code omit the timeout task behavior, as well as the handling of inputs and delivered messages.

The algorithm proceeds in a sequence of *phases*, numbered consecutively starting with 0. Each process attempts to reach a decision at each phase; however, at even-numbered phases, processes are only permitted to decide

<b>Precondition:</b>	initial next-phase transition
$r = 0$	
$v_i = 1$	
<b>Effect:</b>	
<b>broadcast</b> $((0, i))$	
$r := 1$	
<b>Precondition:</b>	initial decision transition
$r = 0$	
$v_i = 0$	
<b>Effect:</b>	
<b>broadcast</b> $((1, i))$	
<b>decide</b> $(0)$	
<b>Precondition:</b>	next-phase transition
$r \geq 1$	
there exists a $j$ such that $(r, j) \in \text{buff}$	
<b>Effect:</b>	
<b>broadcast</b> $((r, i))$	
$r := r + 1$	
<b>Precondition:</b>	decision transition
$r \geq 1$	
for all $j \notin \text{halted}$ , $(r - 1, j) \in \text{buff}$	
there is no $j$ such that $(r, j) \in \text{buff}$	
<b>Effect:</b>	
<b>broadcast</b> $((r + 1, i))$	
<b>decide</b> $(r \bmod 2)$	

FIG. 2. The main agreement algorithm for process  $p_i$ .

on 0, whereas at odd-numbered phases they can only decide on 1. Furthermore, a process is only permitted to decide at a phase  $r$  provided it knows that no process has decided at phase  $r - 1$ . Thus, if any process decides at phase  $r$ , the algorithm ensures that no process can decide at phase  $r + 1$ .

More strongly, the algorithm ensures in this case that every nonfailed, undecided process learns in phase  $r + 2$  that no process has decided at phase  $r + 1$ , and then decides at phase  $r + 2$ . Since  $r + 2$  and  $r$  have the same parity, it follows that all decisions agree.

Validity is ensured by forcing all nonfailed processes to decide at phase 0 in case they all have input 0, and at phase 1 in case they all have input 1. To ensure termination, if a phase  $r$  occurs during which no process fails and such that no process has decided up through (and including) phase  $r$ , then the algorithm ensures that every nonfaulty process will decide no later than phase  $r + 1$ . (Such a phase must occur among the first  $f + 1$  phases.)

The mechanism used by the algorithm to guarantee all of these properties is the following. If a process fails to decide at any phase  $r$ , it broadcasts the number  $r$  before going on to the following phase  $r + 1$ . On the other hand, if a process decides at phase  $r$ , it “skips” broadcasting  $r$  and instead broadcasts

$r + 1$ , before deciding and terminating. In order for a process to decide at phase  $r \geq 1$ , it ensures that it has received the message  $r - 1$  from all nonhalted processes, and no message  $r$  from any process. This ensures that if a process decides at phase  $r$  then no process has decided at phase  $r - 1$ .

Also, if some process  $p$  decides at phase  $r$ , then every undecided process receives the message  $r + 1$  from  $p$  at phase  $r + 1$ , but no message  $r$  from  $p$  (since  $p$  skips sending  $r$ ). This ensures that each undecided and nonfailed process broadcasts  $r + 1$  and goes on to phase  $r + 2$ . Then every undecided, nonfailed process will receive the message  $r + 1$  from all nonfailed processes, and no message  $r + 2$  from any process. It follows that each undecided, nonfailed process decides at phase  $r + 2$ .

The algorithm allows any process having input 0 to decide at phase 0. If all processes have input 1, then no process decides at phase 0. In this case, every nonfailed process broadcasts 0 and no process sends 1, so that every process has its precondition for decision satisfied at phase 1. Validity is thus guaranteed.

For termination, suppose that a phase  $r$  occurs during which no process fails and such that no process decides up to and including phase  $r$ . Then no process sends the message  $r + 1$ , all nonfailed processes send the message  $r$ , and so the preconditions for every process to decide at phase  $r + 1$  are satisfied.

The transitions corresponding to  $comp(i, S)$  events of  $p_i$  are shown in more detail in Figure 2. The code contains preconditions for the various cases; note that, in every state of  $p_i$ , at most one of the four cases has its precondition satisfied. Since  $comp(i, S)$  events are required to be enabled in all states, we use the convention that any state in which none of the four preconditions is satisfied has a “dummy” transition enabled, which causes no changes to the state and no messages to be sent.

A formal proof of correctness appears in Section 5.2.

We indicate why the time required for this algorithm to terminate only involves a single occurrence of the timeout bound  $T \approx Cd + \delta$ , not multiplied by  $f$ . Note that the only transition that occurs because of a timeout is the (noninitial) decision transition. Suppose this transition is ever begun by a process  $p_i$  at a phase  $r$  and no  $(r, j)$  message *ever* arrives at  $p_i$ . Then the timeout can take time  $T$ , but then all nonfailed processes will decide very quickly and terminate the computation. (In fact, all such processes must decide by the same phase  $r$ , since otherwise they would send  $(r, j)$  messages to  $p_i$ .) On the other hand, suppose that, at all phases  $r$  prior to some particular phase  $h$ , whenever a process  $p_i$  begins the decision transition, some  $(r, j)$  message does arrive at  $p_i$ . Then all  $(r, j)$  messages must arrive at  $p_i$  *after* the transition (or the transition would not be enabled). Then we claim that each such phase  $r$  takes only time depending on  $f_r \delta$ , but not on  $T$ . Here,  $f_r$  is the number of failures that occur during a transition where the associated set of *send* actions is  $broadcast((r, j))$  for some  $j$ . This is because each  $(r, j)$  message originates (either directly or via a chain of rebroadcasts) when some process performs a decision transition at phase  $r - 1$ . The length of a shortest such chain can be at most  $f_r + 1$ . This is because a nonfailed process succeeds in communicating its message to everyone. Therefore, the time for phase  $r$  is bounded by  $(f_r + 1)\delta$ , the length of the chain multiplied by the time to deliver each message in the chain.

A careful analysis appears in Section 5.3.



5.2. CORRECTNESS PROOF. When we say that a process *begins* a transition, we mean that the precondition for the transition is satisfied and either the associated  $\text{comp}(i, S)$  step or an associated  $\text{fail}(i, S)$  step is performed. Thus, this does not necessarily mean that the transition described in the code is completed, that is, that the associated  $\text{comp}(i, S)$  step is performed. Note that, for each  $r \geq 0$ ,  $p_i$  begins at most one of the next-phase or decision transitions; we call this *the  $r$ th phase of  $p_i$* . Note also that, if  $p_i$  decides at phase  $r$ , then  $p_i$  completes the decision transition at phase  $r$  and thus sends the message  $(r + 1, i)$  to all processes.

An  $r$ -message is any message of the form  $(r, i)$  for some  $i$ . It follows from the code that an  $r$ -message is sent either at a decision transition at phase  $r - 1$  or at a next-phase transition at phase  $r$ .

We first prove *progress*, that is, that nonfaulty processes do not get “stuck” in a phase: They either decide or advance to the next phase.

LEMMA 5.3. *Let  $r \geq 0$ , and let  $p_i$  be a nonfaulty process. Then  $p_i$  either decides at a phase strictly less than  $r$  or begins a transition at phase  $r$ .*

PROOF. Suppose not. Let  $r$  be the first phase at which a nonfaulty process gets stuck, and let  $p_i$  be a nonfaulty process that does not increase its phase to  $r + 1$ . Since it is not possible for any process to get stuck at phase 0, it must be that  $r \geq 1$ . Process  $p_i$  eventually times out every process  $p_j$  that fails or decides, by Property T2 of the timeout task.

So consider any process  $p_j$  that does not fail or decide. By choice of  $r$ ,  $p_j$  eventually reaches phase  $r$ . Since  $p_j$  does not decide at phase  $r - 1$ , it must have set its phase to  $r$  using a next-phase transition. This implies that  $p_j$  sends an  $(r - 1)$ -message to  $p_i$ . Hence,  $p_i$  eventually receives an  $(r - 1)$ -message from  $p_j$  and uses it to satisfy its waiting condition for  $p_j$ .

Thus,  $p_i$  eventually satisfies its waiting conditions for all  $p_j$  and is able to begin a transition at phase  $r$ , a contradiction to the choice of  $r$  and  $p_i$ .  $\square$

We next give some preliminary lemmas. Some of these lemmas will also be used later in the timing analysis.

LEMMA 5.4. *If  $p_i$  begins a decision transition at phase  $r \geq 0$ , then  $p_i$  sends no  $r$ -messages.*

PROOF. If  $r = 0$ , then by the initial decision transition,  $p_i$  sends no 0-messages. Assume  $r \geq 1$ . If  $p_i$  sends  $r$  at phase  $r - 1$ ,  $p_i$  begins a decision transition at phase  $r - 1$  and does not execute phase  $r$ . Since  $p_i$  begins a decision transition at phase  $r$ , it does not begin a next-phase transition at phase  $r$  and thus does not send an  $r$ -message at phase  $r$ .  $\square$

LEMMA 5.5. *If  $p_i$  decides at phase  $r \geq 0$ , then no process begins a decision transition at phase  $r + 1$ .*

PROOF. Assume, by way of contradiction, that some process  $p_j$  begins a decision transition at phase  $r + 1$ . Then prior to this decision transition, either an  $r$ -message from  $p_i$  is delivered to  $p_j$ , or  $p_j$  adds  $i$  to its set of halted processes. By Lemma 5.4,  $p_i$  does not send any  $r$ -messages, so the only possibility is that  $p_j$  adds  $i$  to *halted*. By the decision transition rule,  $p_i$  succeeds in broadcasting  $r + 1$ . But, by Property T1 of the timeout task, all messages sent by  $p_i$  to  $p_j$  are delivered to  $p_j$  before it adds  $i$  to *halted*. Thus,

an  $(r + 1)$ -message must be delivered to  $p_j$  before it begins the decision transition. But this contradicts the precondition for the decision transition.  $\square$

We next give a definition that will be central to both the correctness proof and the timing analysis. A phase  $r$  is *quiet* if there exists a process  $p_i$  such that no process  $p_j$  sends an  $r$ -message to  $p_i$ .

LEMMA 5.6. *Suppose  $r \geq 1$ . If no process begins a decision transition at phase  $r - 1$ , then phase  $r$  is quiet.*

PROOF. This is true because an earliest sending of an  $r$ -message must occur at a decision transition at phase  $r - 1$ .  $\square$

LEMMA 5.7. *If phase  $r$  is quiet, then all processes either fail or decide by the end of phase  $r$ .*

PROOF. Suppose not; let  $p_i$  be a process that does not fail or decide by the end of phase  $r$ . By Lemma 5.3,  $p_i$  must exit phase  $r$ , so it must perform a next-phase transition at phase  $r$ . Since  $p_i$  does not fail, it broadcasts  $r$ . This contradicts the assumption that phase  $r$  is quiet.  $\square$

LEMMA 5.8. *Assume that some process decides at phase  $r$ . Then phase  $r + 2$  is quiet and all processes either fail or decide no later than phase  $r + 2$ .*

PROOF. By Lemma 5.5, no process begins a decision transition at phase  $r + 1$ . By Lemma 5.6, this implies that phase  $r + 2$  is quiet. So by Lemma 5.7, all either fail or decide no later than phase  $r + 2$ .  $\square$

Now we can prove the agreement property.

LEMMA 5.9. *No two processes decide on different values.*

PROOF. Let  $r$  be the minimal phase at which any process decides, and let  $p_i$  be a process that decides at phase  $r$ . By Lemma 5.5, no process begins a decision transition in phase  $r + 1$ . By Lemma 5.8, all processes either fail or decide no later than phase  $r + 2$ . Since  $r$  is minimal, it follows that all nonfaulty processes decide at phase  $r$  or at phase  $r + 2$ . Since  $r \bmod 2 = (r + 2) \bmod 2$ , they decide on the same value.  $\square$

We next prove the validity property.

LEMMA 5.10. *If  $p_i$  decides  $v$ , then there exists some  $p_j$  that starts with  $v_j = v$ .*

PROOF. Assume by way of contradiction that all processes start with  $v' \neq v$ . If  $v' = 0$ , then all nonfaulty processes decide on 0 at phase 0. If  $v' = 1$ , then no process begins a decision transition at phase 0, so Lemma 5.6 implies that phase 1 is quiet, and so by Lemma 5.7 all nonfaulty processes decide on 1 at phase 1. Either case yields a contradiction.  $\square$

We next argue termination.

LEMMA 5.11. *Any  $f$ -admissible timed execution contains a quiet phase, numbered no larger than  $f + 2$ .*

PROOF. If some process decides at phase  $r \leq f$ , then Lemma 5.8 implies that phase  $r + 2 \leq f + 2$  is quiet. So suppose that no process decides at any phase  $r$  with  $r \leq f$ . Since there are at most  $f$  failures, there must be some phase  $r$ ,  $0 \leq r \leq f$ , at which no process fails; let  $h$  be some such phase. Since

$h \leq f$ , no process decides at phase  $h$ . In fact, no process  $p_i$  begins a decision transition at phase  $h$ , because otherwise  $p_i$  would complete this transition without failing. Therefore, by Lemma 5.6, phase  $h + 1 \leq f + 1$  is quiet.  $\square$

LEMMA 5.12. *In any  $f$ -admissible timed execution of the algorithm, all processes either fail or decide no later than phase  $f + 2$ .*

PROOF. By Lemma 5.11, any  $f$ -admissible timed execution contains a quiet phase, numbered no larger than  $f + 2$ . Then, Lemma 5.7 implies that all processes either fail or decide by phase  $f + 2$ .  $\square$

*Remark 1.* Our algorithm does not require an a priori upper bound on the number of faults. All nonfaulty processes decide no later than phase  $f + 2$ , where  $f$  is the number of faults that actually occur in the execution. In consequence, the algorithm is an “early stopping” algorithm (cf. [Dolev et al., 1990]). If an upper bound  $f$  is known a priori, the algorithm can be modified so that, if  $p_i$  has not yet decided when it makes a next-phase transition from phase  $f + 1$  to phase  $f + 2$ , then  $p_i$  can immediately decide on  $(f + 2) \bmod 2$ . Since  $p_i$  decides no later than the end of phase  $f + 2$ , there is no need to actually execute phase  $f + 2$ .

5.3. TIMING ANALYSIS. Some notation to describe the number of failures is useful.

For each  $r \geq 1$ , denote by  $f_r$  the number of processes whose failure step is a transition during which an  $r$ -message should be broadcast; more precisely, if applying the transition function to the state from which the failure occurs results in sending an  $r$ -message. (This is either a decision transition at phase  $r - 1$  or a next-phase transition at phase  $r$ .) Note that a process has at most one failure step and thus, in all  $f$ -admissible executions,  $\sum_{r \geq 1} f_r \leq f$ .

The key idea behind the upper bound is that, if a phase  $r$  is not quiet, then the time of the phase can be bounded above by a quantity that depends on  $f_r$  but not on  $C$ . Moreover, the time for any phase (in particular, the first quiet phase) is at most  $T \approx Cd + \delta$ . By Lemma 5.7, all nonfaulty processes decide no later than the end of the first quiet phase. Since a quiet phase must occur before too many phases have elapsed, the bound follows.

In more detail, fix an arbitrary  $f$ -admissible timed execution  $\alpha$ . We introduce some notation; all definitions are with respect to  $\alpha$ . For  $r \geq 0$ , define  $t_r$  to be the minimum time  $t$  such that all processes either fail, decide, or perform a phase  $r$  transition no later than time  $t$ . Note that  $t_r \leq t_{r+1}$  for all  $r$ , and  $t_0 \leq s$ , where  $s = \text{start}(\alpha)$ . (Recall that, by definition, every process  $p_i$  has had a computation or failure event by time  $\text{start}(\alpha)$ , which is preceded in  $\alpha$  by the input event at  $p_i$ .) Let  $t_{\text{dec}}$  be the minimum time  $t$  such that all processes either fail or decide no later than time  $t$ . Let  $h$  be the smallest  $r$  such that phase  $r$  is quiet. It follows from Lemma 5.11 that  $h$  exists and  $h \leq f + 2$ .

It is convenient to handle the cases  $h = 0$  and  $f = 0$  separately. If  $h = 0$ , then Lemma 5.7 implies that the algorithm takes time zero. If  $f = 0$ , then since there are no failures it is easy to see that all processes decide no later than the end of phase 2, and that phases 1 and 2 take time at most  $\Delta$  each. The time bound claimed in Theorem 5.1 is at least  $2\Delta$  when  $f = 0$ . Henceforth, we assume that  $h \geq 1$  and  $f \geq 1$ .

We begin with a simple lemma stating that every phase takes at most time  $T$ .

LEMMA 5.13. *For any phase  $r \geq 1$ ,  $t_r \leq t_{r-1} + T$ .*

PROOF. Consider any process  $p_i$  that does not fail or decide by time  $t_{r-1} + T$ . If any process  $p_j$  decides at phase  $r - 1$ , then within time  $\Delta$  after  $p_j$ 's decision transition, (and so by time  $t_{r-1} + \Delta \leq t_{r-1} + T$ ),  $p_i$  receives an  $r$ -message and performs a phase  $r$  next-phase transition.

Now assume that no process decides at phase  $r - 1$ . For any process  $p_j$  that fails or decides at or before its phase  $r - 1$  transition,  $p_i$  puts  $j$  into its *halted* set and takes a subsequent computation or failure step by time  $t_{r-1} + T$ . Also, every process that does not fail or decide at or before its phase  $r - 1$  transition completes a phase  $r - 1$  next-phase transition, in which it sends an  $(r - 1)$ -message; this message is received by  $p_i$  by time  $t_{r-1} + \Delta \leq t_{r-1} + T$ . Since no process decides at phase  $r - 1$ ,  $p_i$  receives no  $r$ -messages. It follows that  $p_i$  performs a phase  $r$  decision transition by time  $t_{r-1} + T$ .

Applying the preceding argument to all  $p_i$ , we conclude that  $t_r \leq t_{r-1} + T$ .  $\square$

The next lemma is the key to the upper bound. It says that the time required by a nonquiet phase is short (in particular, independent of  $C$ ). The reason is that the length of such a phase is bounded by the time to deliver a chain of messages of length one more than the number of failures at that phase. The details follow.

LEMMA 5.14. *For any  $r$  with  $1 \leq r \leq h - 1$ ,  $t_r \leq t_{r-1} + \Delta(f_r + 1)$ .*

PROOF. Let  $p_i$  be an arbitrary process. Assume that  $p_i$  does not fail or decide by time  $t_{r-1} + \Delta(f_r + 1)$ . Since phase  $r$  is not quiet, some process sends an  $r$ -message to  $p_i$ . By inspection of the algorithm, there must be a sequence  $i_0, \dots, i_k$  of distinct process indices with  $i_k = i$ , such that  $p_{i_0}$  sends an  $r$ -message to  $p_{i_1}$  while performing a decision transition at phase  $r - 1$  and, for  $1 \leq j \leq k - 1$ ,  $p_{i_j}$  sends an  $r$ -message to  $p_{i_{j+1}}$  while performing a next-phase transition at phase  $r$ . Choosing the sequence of process indices so that  $k$  is minimized, it follows that, for  $0 \leq j \leq k - 2$ ,  $p_{i_j}$  fails during the broadcast of the  $r$ -message. For if  $p_{i_j}$  does not fail, then it sends an  $r$ -message to  $p_i$ , so  $i_0, \dots, i_j, i$  would give a path of length less than  $k$  from  $p_{i_0}$  to  $p_i$ .

By definition of  $f_r$ , we have  $k - 1 \leq f_r$ . Since  $p_{i_0}$  sends the  $r$ -message no later than time  $t_{r-1}$ , and  $p_{i_1}, \dots, p_{i_k}$  enter phase  $r$  no later than time  $t_{r-1}$ , it follows that  $p_i$  receives the  $r$ -message and satisfies the precondition for a next-phase transition no later than time  $t_{r-1} + k\Delta \leq t_{r-1} + (f_r + 1)\Delta$ . Since, by assumption,  $p_i$  does not fail or decide by time  $t_{r-1} + (f_r + 1)\Delta$ ,  $p_i$  performs a phase  $r$  next-phase transition by this time.  $\square$

Now by induction, we have:

COROLLARY 5.15. *For every  $r$  with  $1 \leq r \leq h - 1$ ,  $t_r \leq \Delta \cdot \sum_{i=1}^r (f_i + 1) + s$ .*

At this point, we can give a simple proof of an upper bound result that is slightly weaker than the one claimed in Theorem 5.1. We include this result here in order to give the reader an intuition why the bound takes the general form it does (with the timeout bound  $T$  appearing only once).

THEOREM 5.16. *There is an algorithm to solve the agreement problem for  $f$  faults within time  $(2f + 1)\Delta + T$ .*

Again assuming  $c_2 \ll \delta$ , this bound is approximately  $(2f + 2)\delta + Cd$ .

PROOF. By Lemma 5.7, we have  $t_{\text{dec}} \leq t_h$ . Lemma 5.13 implies that  $t_r \leq t_{r-1} + T$  for any phase  $r$ . Therefore,  $t_{\text{dec}} \leq t_{h-1} + T$ . Now

$$\begin{aligned}
 t_{\text{dec}} &\leq t_{h-1} + T \\
 &\leq \Delta \cdot \sum_{i=1}^{h-1} (f_i + 1) + T + s \quad \text{by Corollary 5.15,} \\
 &\leq (f + (h - 1))\Delta + T + s \\
 &\leq (2f + 1)\Delta + T + s \quad \text{since } h \leq f + 2. \quad \square
 \end{aligned}$$

Now we carry out the finer analysis needed to get the smaller bound given in Theorem 5.1. In the case that  $\delta$  is at least (approximately)  $d/2$ , the smaller bound is close (within  $O(c_2(C + f))$ ) to the actual worst-case running time of the algorithm; see Remark 2 below. The better bound is obtained by considering the latest time at which a failure occurs. If this time is not too large, then a better bound can be obtained since the time  $T$  taken by the timeout task can then be measured starting from the time of the latest failure. Let  $t_{\text{last}}$  be the maximum time such that  $t_{\text{last}} \leq t_{h-1}$  and such that some process has a failure event at time  $t_{\text{last}}$ . If no process has a failure event at a time  $\leq t_{h-1}$ , then take  $t_{\text{last}} = -T$  (so that the following lemma will be valid in this case). We begin with an upper bound on  $t_{\text{dec}}$  that may be smaller than the bound  $t_{h-1} + T$  used in the proof of Theorem 5.16.

LEMMA 5.17.  $t_{\text{dec}} \leq \max\{t_{h-1} + \Delta, t_{\text{last}} + T\}$ .

PROOF. By Lemma 5.7,  $t_{\text{dec}} \leq t_h$  so it is sufficient to bound  $t_h$ . Let  $p_i$  be a process that does not fail or decide before time  $t_{\text{max}} = \max\{t_{h-1} + \Delta, t_{\text{last}} + T\}$ . Let  $p_j$  be an arbitrary process. We show that, by time  $t_{\text{max}}$ , either  $j$  is in  $p_i$ 's *halted* set or  $p_i$  receives an  $(h - 1)$ -message or an  $h$ -message from  $p_j$ . Therefore, by time  $t_{\text{max}}$ ,  $p_i$  performs a phase  $h$  transition.

If  $p_j$  fails at time  $t$  where  $t \leq t_{h-1}$ , then  $t \leq t_{\text{last}}$ , so  $p_i$  adds  $j$  to its *halted* set no later than time  $t_{\text{last}} + T$  (by Property T2 of the timeout task). In the remaining cases, assume that  $p_j$  does not fail at a time  $t \leq t_{h-1}$ .

Suppose that  $p_j$  performs a transition at phase  $h - 1$ . Since  $p_j$  does not fail at this transition,  $p_j$  sends either an  $(h - 1)$ -message or an  $h$ -message to  $p_i$ . Since the sending is done no later than time  $t_{h-1}$ ,  $p_i$  receives the message no later than time  $t_{h-1} + \Delta$ .

The only other possibility is that  $p_j$  decides at some phase  $r \leq h - 2$ . Since  $p_i$  does not fail or decide by the end of phase  $h - 1$ , it follows from Lemma 5.8 that  $p_j$  does not decide at any phase  $r \leq h - 3$ . Therefore,  $p_j$  decides at phase  $h - 2$  and broadcasts an  $(h - 1)$ -message. As in the previous case, this message is received by  $p_i$  no later than time  $t_{h-2} + \Delta \leq t_{h-1} + \Delta$ .  $\square$

We now use Lemma 5.17 to bound  $t_{\text{dec}}$ .

LEMMA 5.18.  $t_{\text{dec}} \leq \max\{(2f + 2)\Delta, (2f - 1)\Delta + T\} + s$ .

PROOF. We consider three cases.



Case 1.  $h \leq f$ .

Since  $t_{\text{dec}} \leq t_{h-1} + T$ , Corollary 5.15 gives

$$\begin{aligned} t_{\text{dec}} &\leq t_{h-1} + T \\ &\leq \Delta \cdot \sum_{i=1}^{h-1} (f_i + 1) + T + s \\ &\leq (f + (h - 1))\Delta + T + s \\ &\leq (2f - 1)\Delta + T + s \quad \text{since } h \leq f. \end{aligned}$$

Case 2.  $f + 1 \leq h \leq f + 2$  and  $t_{\text{last}} \leq t_{f-1}$ .

First, since  $f - 1 < h - 1$  we have

$$t_{\text{last}} \leq t_{f-1} \leq \Delta \cdot \sum_{i=1}^{f-1} (f_i + 1) + s \leq (2f - 1)\Delta + s.$$

Since  $h - 1 \leq f + 1$  we have

$$t_{h-1} \leq \Delta \cdot \sum_{i=1}^{h-1} (f_i + 1) + s \leq (2f + 1)\Delta + s.$$

Substituting these bounds for  $t_{\text{last}}$  and  $t_{h-1}$  into Lemma 5.17 gives

$$\begin{aligned} t_{\text{dec}} &\leq \max\{(2f + 1)\Delta + s + \Delta, (2f - 1)\Delta + s + T\} \\ &= \max\{(2f + 2)\Delta, (2f - 1)\Delta + T\} + s. \end{aligned}$$

Case 3.  $f + 1 \leq h \leq f + 2$  and  $t_{\text{last}} > t_{f-1}$ .

CLAIM 5.19.  $f_r > 0$  for  $1 \leq r \leq f - 1$ .

PROOF. Suppose that  $f_r = 0$  for some  $r \leq f - 1$ . Since phase  $r$  is not quiet, some process sends an  $r$ -message, and the earliest sending of an  $r$ -message must be at a decision transition at phase  $r - 1$ . Since  $f_r = 0$  means that there are no failures during a broadcast of an  $r$ -message, it follows that some process decides at phase  $r - 1$ . By Lemma 5.8, phase  $r + 1$  is quiet. Since  $r + 1 \leq f$ , this contradicts the assumption that phase  $h \geq f + 1$  is the first quiet phase.  $\square$

Since phase  $f$  is not quiet, an  $f$ -message is sent by some process. Let  $p$  be a process that sends an  $f$ -message at the earliest time. Therefore,  $p$  sends the  $f$ -message while performing a decision transition at phase  $f - 1$ , and this occurs no later than time  $t_{f-1}$ .

We first argue that  $p$  decides at phase  $f - 1$ . If not, then  $p$  fails no later than time  $t_{f-1}$  while broadcasting an  $f$ -message. Since  $f_r > 0$  for  $r \leq f - 1$ , the remaining  $f - 1$  failures occur while some process is broadcasting an  $r$ -message for each  $r$  with  $1 \leq r \leq f - 1$ . Since these remaining failures occur at phases numbered at most  $f - 1$ , it follows that all failures occur no later than time  $t_{f-1}$ . This contradicts the assumption that  $t_{\text{last}} > t_{f-1}$ .

Since  $p$  decides at phase  $f - 1$ ,  $h = f + 1$  by Lemma 5.8, and  $p$  broadcasts an  $f$ -message no later than time  $t_{f-1}$ . Therefore

$$t_{h-1} = t_f \leq t_{f-1} + \Delta. \quad (1)$$

The final ingredient for this case is the observation that

$$\sum_{i=1}^{f-1} f_i \leq f - 1. \quad (2)$$

Otherwise, all failures occur during the broadcast of  $r$ -messages for  $1 \leq r \leq f - 1$ ; as argued above, this contradicts the assumption that  $t_{\text{last}} > t_{f-1}$ .

Finally, we have

$$\begin{aligned} t_{\text{dec}} &\leq t_{h-1} + T \\ &\leq t_{f-1} + \Delta + T && \text{by (1)} \\ &\leq \Delta \cdot \sum_{i=1}^{f-1} (f_i + 1) + s + \Delta + T \\ &\leq ((f - 1) + (f - 1))\Delta + s + \Delta + T && \text{by (2)} \\ &= (2f - 1)\Delta + T + s. \end{aligned} \quad \square$$

Since the upper bound of Lemma 5.18 can be written as  $(2f - 1)\Delta + \max\{T, 3\Delta\} + s$ , the proof of Theorem 5.1 is complete.

*Remark 2.* It is possible to construct an execution of the algorithm that takes time approximately  $2f\delta + Cd$ , assuming  $1 \leq f \leq n - 2$ ,  $C \geq 2$ , and that  $\delta$  is at least  $d/2$  plus a small multiple of  $c_2$ . Some hints toward the construction follow. To simplify the description, all times are approximate (to within  $O(c_2 f)$ ). Process  $p_1$  has initial value 0 and the others have initial value 1. For  $0 \leq r \leq f - 2$ , process  $p_{r+1}$  performs a phase- $r$  decision transition at time  $2r\delta$ . This transition is a failure transition during which an  $(r + 1)$ -message is sent only to  $p_n$ , and  $p_n$  receives this message and rebroadcasts it at time  $2r\delta + \delta$ . Now during phase  $r + 1$ , process  $p_{r+2}$  times-out  $p_{r+1}$  at time  $2(r + 1)\delta$  before receiving the  $(r + 1)$ -message from  $p_n$  and it performs a decision transition, whereas  $p_{r+3}, \dots, p_{n-1}$  receive the  $(r + 1)$ -message from  $p_n$  at time  $2(r + 1)\delta$  before timing-out  $p_{r+1}$  so they perform a next-phase transition, so the pattern can continue. Then  $p_f$  completes a phase- $(f - 1)$  decision transition at time  $2(f - 1)\delta$  and broadcasts an  $f$ -message. Then  $p_n$  fails at time  $2(f - 1)\delta + \delta$  and does not rebroadcast the  $f$ -message. An additional timeout time of  $Cd + \delta$  then elapses between the time of this last failure and the time  $2f\delta + Cd$  when  $p_{n-1}$  can decide.

*Remark 3.* The agreement algorithm has high-message complexity. This is due mainly to the timeout task where every process broadcasts a message at every step—the main task sends a total of  $O(n^2 f)$  messages, since each process broadcasts a message at each phase transition. (Each message has length  $O(\log n)$  bits.) An obvious approach for decreasing the message complexity of the timeout task is to broadcast the *alive* message once every  $k$  steps for some

$k \geq 2$ . Of course, the maximum value of the counters must then be adjusted upward, and the timeout bound  $T$  increases accordingly.

For the case of synchronized start, another approach is to dispense with the timeout task completely and build special timeout mechanisms into the main algorithm. Specifically, whenever  $p_i$  makes a next-phase transition from phase  $r - 1$  to phase  $r$ , it initializes a counter  $counter(j)$  for each  $p_j$ . Each counter  $counter(j)$  is incremented at each step until either (i)  $p_i$  receives an  $r$ -message (causing it to perform a next-phase transition), (ii) the message  $(r - 1, j)$  is found in *buff*, or (iii)  $counter(j)$  reaches  $\lfloor 2D/c_1 \rfloor + 1$ . In case (iii),  $p_i$  adds  $j$  to *halted*. The modified algorithm is correct since, whenever  $p_i$  broadcasts an  $(r - 1)$ -message during a next-phase transition at phase  $r - 1$ , it should receive either an  $(r - 1)$ -message or an  $r$ -message from every nonfaulty nondecided process within time  $2D$ . The modified algorithm sends a total of  $O(n^2f)$  messages. Again, each message has length  $O(\log n)$  bits. By a timing analysis similar to that of Theorem 5.16, an upper bound of  $(2f + 1)\Delta + 2CD + c_2 \approx (2f + 1)\delta + 2Cd$  can be shown.

**5.4. EXTENSION TO MULTIPLE VALUES.** In this section, we discuss how to modify the algorithm to handle an arbitrary value set  $\mathcal{V}$ . This is done by running  $n$  single-source algorithms in parallel. In the *single-source agreement* problem, a single process  $p_i$ , the *source*, starts with an initial value from  $\mathcal{V}$ . Shortly, we will describe an algorithm for the single-source problem with the following properties. Let  $\perp$  be a distinguished *default value* in  $\mathcal{V}$ . Suppose that the source has initial value  $v$ . Then all nonfaulty processes decide on either  $v$  or  $\perp$ , and all decide the same; moreover, if the source is nonfaulty, then all nonfaulty processes decide on  $v$ . To solve the general agreement problem, run  $n$  single-source algorithms,  $A_1, \dots, A_n$ , in parallel with  $p_i$  being the source in  $A_i$ . When some process  $p_j$  has reached a decision  $w_i$  in  $A_i$  for all  $i$ , it decides on  $w_k$  where  $k$  is the least integer such that  $w_k \neq \perp$ . (Such a  $k$  must exist, since  $w_j \neq \perp$ .)

To describe a solution to the single-source problem, we refer to the algorithm of Figure 2 as the *binary algorithm*. Let  $p_i$  be the source, and let  $v_i \in \mathcal{V}$  be the initial value of  $p_i$ . Initially,  $p_i$  begins the binary algorithm as though it has initial value 0, and the other processes begin with value 1. During phase 0,  $p_i$  broadcasts the message  $(v_i, (1, i))$ ; that is, it sends the message  $(1, i)$  that the binary algorithm would send, with the value  $v_i$  piggybacked. After this broadcast,  $p_i$  decides  $v_i$ . Any process that receives this message during phase 1 remembers  $v_i$ , broadcasts  $(v_i, (1, i))$ , and otherwise acts in the binary algorithm as though the message  $(1, i)$  had been received. The binary algorithm is then run to completion. If a process decides 0 (respectively, 1) in the binary algorithm, it decides  $v_i$  (respectively,  $\perp$ ) in the single-source algorithm. (The analysis below shows that, if  $p_j$  decides 0 in the binary algorithm, then  $p_j$  receives  $v_i$  during phase 1.)

To argue correctness, first consider the case that the source  $p_i$  is nonfaulty. It is easy to see in this case that all nonfaulty processes (except the source) decide 0 at phase 2 in the binary algorithm, so all decide  $v_i$  in the single-source algorithm. If  $p_i$  is faulty, let  $R$  be the set of processes that receive  $(v_i, (1, i))$  during phase 1. Any process not in  $R$  either fails or performs a decision transition at phase 1. If any such process decides, then all nonfaulty processes

decide 1. If all processes that are not in  $R$  fail before deciding, then any process  $p_j$  that does decide is in  $R$ , so  $p_j$  receives  $v_i$  during phase 1.

## 6. The Lower Bound

In this section, we prove our lower bound of  $(f - 1)d + Cd$  on the time to reach agreement in the timing-based model. The proof requires four steps and employs techniques used elsewhere in proving lower bounds and impossibility results in the rounds model, the completely asynchronous model, and the timing-based model.

The first step is an adaptation of the proof showing that  $f + 1$  rounds are necessary for Byzantine agreement in the rounds model.<sup>7</sup> As we shall see, this adaptation yields the existence of two “long” (i.e., taking time at least  $(f - 1)d$ ) timed execution prefixes,  $\alpha_0$  and  $\alpha_1$ , each having only  $f - 1$  faults, distinguishable only to one process, and each extendible to a timed execution with a different decision value. (This is done in Lemma 6.1.)

The second step mimics a key lemma in the proof that agreement is impossible in asynchronous systems [Dolev et al., 1987; Fischer et al., 1985]. In this step, it is shown that at least one of  $\alpha_0$  and  $\alpha_1$  is “bivalent,” in that it has two possible extensions with no additional failures, each yielding a different decision value and in each of which processes take steps as quickly as possible. In showing bivalence, we also use an “execution retiming” technique [Attiya and Lynch, 1989]. (This is done in Corollary 6.3.)

The third step extends the bivalent timed execution prefix to a bivalent prefix, having at most  $f - 1$  faults, which is “maximal,” that is, all its extensions are univalent. (This is done in Lemma 6.4.) The fourth and last step exploits the one remaining fault, via another retiming argument, to show that, after this maximal bivalent timed execution prefix, at least one “long timeout” (taking time at least  $Cd$ ) is necessary. (This is done in Theorem 6.5.)

We assume throughout this section that  $c_1 \leq d$ ,  $\delta = d$ , and  $f \geq 1$ .

**6.1. SYNCHRONOUS TIMED EXECUTIONS.** Our lower bound arguments for algorithms in the timing-based model will be based on a subset of the timed executions that we call “synchronous.” We define these in this subsection.

We think of a synchronous timed execution as a sequence of “blocks”; each block is composed of a sequence of message deliveries followed by a sequence of process steps; all the process steps in one block occur at the same time, and each block contains exactly one (computation or failure) step by each process. More precisely, we say that a timed execution is *synchronous*, provided that there is a monotone increasing sequence of times,  $t_0, t_1, \dots$ , such that  $t_0 = 0$  and the following conditions are satisfied.

- (1) Exactly one input event occurs at each process, and it occurs at time 0.
- (2) Each computation and failure event occurs at time  $t_i$ , for some  $i$ . At each time  $t_i$ , there is exactly one computation or failure event for each process, and these events occur in order of process indices.

<sup>7</sup>See Coan and Dwork [1991]; DeMillo et al. [1982]; Dolev and Strong [1983]; Dwork and Moses [1990]; Fischer and Lynch [1982]; Hadzilacos [1984]; Lamport and Fischer [1982]; Merritt [1985].

- (3) All input events precede all computation and failure events that occur at time 0.
- (4) All message delivery events that occur at a time  $t_i$  precede all computation and failure events that occur at the same time.

A *block* in a synchronous timed execution can then be identified with the portion of the execution occurring at times in the interval  $(t_i, t_{i+1}]$  for any particular  $i$ . A (finite) timed execution prefix is said to be *synchronous*, provided that it is a prefix of a synchronous timed execution and it ends with a computation or failure step of process  $p_n$ .

Now suppose that  $\alpha$  is a synchronous timed execution prefix. If  $\gamma = \alpha\beta$  is a synchronous timed execution or a synchronous timed execution prefix, we say that  $\gamma$  is a *failure-free extension* (or simply *ff-extension*) of  $\alpha$  if no failures occur in  $\beta$ . We say that  $\gamma$  is a *fast extension* of  $\alpha$  if the times for computation and failure steps in  $\gamma$  that are greater than  $t_{\text{end}}(\alpha)$  are exactly all the times that are of the form  $t_{\text{end}}(\alpha)$  plus a positive multiple of  $c_1$ . Similarly,  $\gamma$  is a *slow extension* of  $\alpha$  if the computation and failure step times are all those of the form  $t_{\text{end}}(\alpha)$  plus a positive multiple of  $c_2$ .

Intuitively, in a fast extension, processes take steps as fast as possible, while in a slow extension, processes take steps as slow as possible. Fast extensions are important since they can be retimed to become slow (and take more time); this fact is crucial for the proof of Lemma 6.2 below, which is the key to the lower bound result.

**6.2. EXISTENCE OF LONG PREFIXES.** For the first step, we show the existence of the two long timed execution prefixes mentioned above. Since we do this by adapting a proof from the rounds model, it is useful for us to restrict attention to a subclass of the synchronous timed executions that look more like executions of the rounds model. In particular, we consider timed executions in which messages are delivered in batches at times that are positive multiples of  $d$ . Also, although step time is irrelevant here, we say (to be specific) that processes take steps at every multiple of  $c_1$ , starting with 0. Formally, we define the *uniform* timed executions to be those synchronous timed executions in which

- (1) for every integer  $r \geq 1$ , any message that is sent at time  $t$ , with  $(r - 1)d \leq t < rd$ , is delivered at time  $rd$ , and
- (2) each step time  $t_i$  is equal to  $ic_1$ .

Also, the *uniform timed execution prefixes* are defined to be the timed execution prefixes that are prefixes of uniform timed executions and end with a computation or failure event of  $p_n$ .

Uniform timed executions are similar to executions in the rounds model. For example, if  $c_1 = d$ , then there is a direct correspondence between the two. In general uniform executions, however, a process may take several steps (and send at several different times) within each round of message exchange.

The basic lower bound result for agreement in the rounds model asserts that, for  $f \leq n - 2$ , agreement in the presence of stopping failures requires  $f + 1$  rounds [Coan and Dwork, 1991; Dwork and Moses, 1990; Hadzilacos, 1984; Lamport and Fischer, 1982; Merritt, 1985]. The proof of this result contains a key lemma that shows, loosely speaking, that, for any agreement algorithm, all



execution prefixes with at most  $f$  rounds in which at most one process fails in each round are *similar*. Two execution prefixes are *directly similar* if some nonfaulty process cannot “distinguish between” them. The similarity relation is the transitive closure of the direct similarity relation.

By redefining “directly similar” so that two execution prefixes are directly similar if *at most one* process *can* distinguish between them, and redefining “similar” accordingly, it is easy to modify this standard proof to apply to our uniform timed executions and to yield a slightly stronger conclusion. In this way, we obtain Lemma 6.1.

We define two timed execution prefixes,  $\alpha_0$  and  $\alpha_1$ , with  $t_{\text{end}} = t_{\text{end}}(\alpha_0) = t_{\text{end}}(\alpha_1)$ , to be *indistinguishable* to process  $p_i$  provided that (a) the sequence of timed events occurring at  $p_i$  and the sequence of intervening local states of  $p_i$  are identical in  $\alpha_0$  and  $\alpha_1$ , with the exception that corresponding *fail* events of  $p_i$  in the two event sequences can send different sets of messages, and (b) the messages that are sent to  $p_i$  strictly before time  $t_{\text{end}}$ , together with their senders and sending times, are identical in  $\alpha_0$  and  $\alpha_1$ . The sequences  $\alpha_0$  and  $\alpha_1$  are said to be *distinguishable* to  $p_i$  if they are not indistinguishable to  $p_i$ .

LEMMA 6.1. *Let  $A$  be an  $n$ -process algorithm in the timing-based model that solves the agreement problem for  $f \leq n - 1$  faults. Let  $k$  be a nonnegative integer,  $k \leq f - 1$ . Then there are two uniform timed execution prefixes,  $\alpha_0$  and  $\alpha_1$ , satisfying the following conditions:*

- (1)  $t_{\text{end}}(\alpha_j) = \lceil kd/c_1 \rceil c_1$ , for  $j = 0, 1$ ,<sup>8</sup>
- (2) *There is a fast ff-extension of  $\alpha_j$  in which some process decides  $j$ , for  $j = 0, 1$ ,*
- (3) *If  $F_j$  is the set of processes that are faulty in  $\alpha_j$ ,  $j = 0, 1$ , then  $|F_0 \cup F_1| \leq k$ , and*
- (4) *There is at most one process to which  $\alpha_0$  and  $\alpha_1$  are distinguishable.*

Note that Lemma 6.1 is stated so as to produce *fast failure-free* extensions because this property is needed in the proof of Corollary 6.3.

PROOF. (SKETCH). For those who are familiar with the earlier proofs (e.g., [Coan, 1987; Coan and Dwork, 1991]: The proof involves constructing a “chain” of timed execution prefixes. Each pair of consecutive prefixes either (a) have identical sets of failed processes and differ only in the presence or absence of one particular message  $m$  sent by a faulty process  $p_i$  to a process  $p_j$ ; moreover,  $p_j$  does not send any messages (in either prefix) at or after the delivery time of  $m$  and strictly prior to  $t_{\text{end}}$ , or (b) differ only in that one process that, in both prefixes, sends all its messages at some time  $t_i$  but none thereafter does a failure transition at time  $t_i$  in one case and at  $t_{i+1}$  in the other case, or (c) differ only in that one process that sends all its messages at time  $t_{\text{end}}$  does a failure transition at time  $t_{\text{end}}$  in one prefix and does not fail in the other prefix, or (d) differ only in the initial value of one process that fails at time 0 and sends no messages.  $\square$

6.3. EXISTENCE OF A LONG BIVALENT PREFIX. For the second step, we show that, under the assumption that agreement can be reached in time strictly less than  $(f - 1)d + Cd$ , both decisions are still possible after at least one of

<sup>8</sup>Note that the time  $\lceil kd/c_1 \rceil c_1$  is the least multiple of  $c_1$  greater than or equal to  $kd$ .

$\alpha_0, \alpha_1$ , where  $\alpha_0, \alpha_1$  are the uniform timed executions given by Lemma 6.1. In order to do this, we need to formalize the notion that “both decisions are still possible” after a prefix. Let  $\alpha$  be a synchronous timed execution prefix.

We say that a value  $v \in \{0, 1\}$  is *fast failure-free-reachable* (or just *fast ff-reachable*) from  $\alpha$  if there is a synchronous fast failure-free extension  $\gamma$  of  $\alpha$  such that some process decides  $v$  in  $\gamma$ . We say that  $\alpha$  is *0-valent* if only 0 is fast ff-reachable from  $\alpha$ , and *1-valent* if only 1 is fast ff-reachable. We say that  $\alpha$  is *univalent* if it is either 0-valent or 1-valent, and that  $\alpha$  is *bivalent* if both 0 and 1 are fast ff-reachable from  $\alpha$ .<sup>9</sup>

The next lemma is the key for completing the proof of the lower bound. It shows that there cannot be two “long” execution prefixes (i.e., prefixes that end at a “late” time) that have opposite valence, that do not contain many faults, and that are distinguishable to at most one process.

LEMMA 6.2. *Let  $A$  be an algorithm in the timing-based model that solves the agreement problem for  $f \leq n - 1$  faults within time strictly less than  $t + Cd$ , for some  $t$ . Then there do not exist two synchronous timed execution prefixes,  $\alpha_0$  and  $\alpha_1$ , satisfying the following properties:*

- (1)  $t_{\text{end}}(\alpha_0) = t_{\text{end}}(\alpha_1) \geq t$ ,
- (2)  $\alpha_j$  is  $j$ -valent,  $j = 0, 1$ ,
- (3) if  $F_j$  is the set of processes that are faulty in  $\alpha_j$ ,  $j = 0, 1$ , then  $|F_0 \cup F_1| \leq f - 1$ , and
- (4) there is at most one process to which  $\alpha_0$  and  $\alpha_1$  are distinguishable.

PROOF. The intuition behind the proof is that, if the process, say  $p$ , which can distinguish the two execution prefixes fails, then any two similar extensions of  $\alpha_0$  and  $\alpha_1$  lead to the same decision, which contradicts the assumption that  $\alpha_0$  and  $\alpha_1$  have different valence. The contradiction is resolved if the failure of  $p$  is detected, but this can take an additional time  $Cd$  after time  $t$ . Very informally, the proof proceeds by assuming that  $\alpha_0$  and  $\alpha_1$  exist. We have  $p$  fail at time  $t$ . Then two similar slow extensions of the two execution prefixes, with no additional failures, must lead to the same decision value. By retiming the slow extensions to be fast, reviving  $p$ , but delaying certain messages of  $p$  as much as possible, we obtain two fast ff-extensions that lead to the same decision. This contradicts the assumption that  $\alpha_0$  and  $\alpha_1$  have different valence. An important fact is that, if a slow extension taking time less than  $Cd$  is retimed to be fast, the fast extension takes time less than  $d$ . The details follow.

Suppose, by way of contradiction, that such prefixes  $\alpha_0$  and  $\alpha_1$  exist. Let  $F$  be the union of  $F_0, F_1$ , and the set (of size at most 1) of processes to which  $\alpha_0$  and  $\alpha_1$  are distinguishable; note that  $|F| \leq f$ . For  $j = 0, 1$ , let  $\alpha'_j$  be a synchronous timed execution prefix that is identical to  $\alpha_j$  except that each  $p_i \in F$  does a failure step in which it sends no messages at time  $t_{\text{end}}$  if it has not failed previously in  $\alpha_j$ . Let  $\gamma_0$  be a slow ff-extension of  $\alpha'_0$ . Let  $\gamma_1$  be constructed in a similar way from  $\alpha'_1$ , subject to the additional condition that

<sup>9</sup>The terminology is derived from that of Fischer et al. [1985], although the definitions are not exactly equivalent.

the portion of  $\gamma_1$  after time  $t_{\text{end}}$  is identical to the portion of  $\gamma_0$  after time  $t_{\text{end}}$ . This is possible since  $\alpha'_0$  and  $\alpha'_1$  are indistinguishable to all processes other than those in  $F$ , and moreover all messages in transit to these processes at time  $t_{\text{end}}$  are the same in  $\alpha'_0$  and  $\alpha'_1$ .

Since  $|F| \leq f$ , it follows that each of  $\gamma_0$  and  $\gamma_1$  is  $f$ -admissible. Since  $t_{\text{end}} \geq t$  and the algorithm decides before time  $t + Cd$ , all the nonfaulty processes, that is, those processes not in  $F$ , decide in each of  $\gamma_0$  and  $\gamma_1$  strictly before time  $t_{\text{end}} + Cd$ . Since  $\gamma_0$  and  $\gamma_1$  are indistinguishable to all processes other than those in  $F$ , they have the same decision value  $v$ . Fix  $j = 1 - v$ . (This makes sense because  $v \in \{0, 1\}$ .)

Let  $\gamma'_j$  be a retiming of  $\gamma_j$  that keeps the times of all events up to and including  $t_{\text{end}}$  the same, and that causes every event that occurs at time  $t_{\text{end}} + u$  in  $\gamma_j$ , for  $u > 0$ , to occur at time  $t_{\text{end}} + u/C$  in  $\gamma'_j$ . Then all processes not in  $F$  decide  $v$  in  $\gamma'_j$ , strictly before time  $t_{\text{end}} + d$ .

Now let  $\gamma''_j$  be a fast ff-extension of  $\alpha_j$  in which any messages sent by processes in  $F$  at times greater than or equal to  $t_{\text{end}}$  take time exactly  $d$  to be delivered, and such that  $\gamma''_j$  looks exactly like  $\gamma'_j$  to all processes except those in  $F$  at times before  $t_{\text{end}} + d$ . Since the processes not in  $F$  cannot tell the difference between  $\gamma''_j$  and  $\gamma'_j$  strictly before time  $t_{\text{end}} + d$ , all processes not in  $F$  must decide  $v$  in  $\gamma''_j$ .

But since  $\gamma''_j$  is a fast ff-extension of  $\alpha_j$  and  $\alpha_j$  is  $j$ -valent, the processes that are nonfaulty in  $\gamma''_j$  must decide  $j$  in  $\gamma''_j$ . Since the processes not in  $F$  are nonfaulty in  $\gamma''_j$ , this is a contradiction.  $\square$

**COROLLARY 6.3.** *Let  $A$  be an algorithm in the timing-based model that solves the agreement problem for  $f \leq n - 1$  faults within time strictly less than  $(f - 1)d + Cd$ . Then there is an  $(f - 1)$ -admissible synchronous timed execution prefix  $\alpha$  such that the following conditions hold:*

- (1)  $t_{\text{end}}(\alpha) = \lceil (f - 1)d/c_1 \rceil c_1$ , and
- (2)  $\alpha$  is bivalent.

**PROOF.** Let  $\alpha_0$  and  $\alpha_1$  be obtained by setting  $k = f - 1$  in Lemma 6.1. We show that at least one of  $\alpha_0$  and  $\alpha_1$  has the required properties. All properties except the bivalence are immediate, so we must show that at least one of  $\alpha_0$  and  $\alpha_1$  is bivalent. We proceed by contradiction. Assume that neither of  $\alpha_0$  and  $\alpha_1$  is bivalent. Then, for  $j = 0, 1$ , since a decision of  $j$  is possible in a fast ff-extension of  $\alpha_j$  (by Lemma 6.1), it must be that  $\alpha_j$  is  $j$ -valent. But then  $\alpha_0$  and  $\alpha_1$  satisfy all the conditions described in the statement of Lemma 6.2, where  $t = (f - 1)d$ . Lemma 6.2 then yields a contradiction.  $\square$

**6.4. EXISTENCE OF A LONG MAXIMAL BIVALENT PREFIX.** For the third step, we construct a “maximal” finite bivalent extension  $\alpha'$  of the bivalent timed execution prefix obtained in the previous corollary. Roughly speaking, the end of  $\alpha'$  is a branch point from which both decisions are still fast ff-reachable and such that at the next step time in any fast ff-extension of  $\alpha'$  the decision must be determined.

**LEMMA 6.4.** *Let  $A$  be an algorithm in the timing-based model that solves the agreement problem for  $f \leq n - 1$  faults within time strictly less than  $(f - 1)d +$*

*Cd. Then  $A$  has an  $(f - 1)$ -admissible synchronous timed execution prefix  $\alpha'$  such that*

- (1)  $t_{\text{end}}(\alpha') \geq (f - 1)d$ ,
- (2)  $\alpha'$  is bivalent, and
- (3) *there are two fast ff-extensions of  $\alpha'$ , namely,  $\beta_j$  for  $j = 0, 1$ , such that*
  - (a)  $\beta_j$  is an extension of  $\alpha'$  by exactly one block,  $j = 0, 1$ ,
  - (b)  $\beta_j$  is  $j$ -valent,  $j = 0, 1$ , and
  - (c)  $\beta_0$  and  $\beta_1$  are indistinguishable to all but at most one process.

PROOF. By Corollary 6.3,  $A$  has a  $(f - 1)$ -admissible synchronous timed execution prefix  $\alpha$  satisfying the following properties:

- (1)  $t_{\text{end}}(\alpha) = \lceil (f - 1)d/c_1 \rceil c_1$ , and
- (2)  $\alpha$  is bivalent.

Let  $\Gamma$  be the set of finite bivalent fast ff-extensions of  $\alpha$ . Each such extension must have its final time strictly less than  $(f - 1)d + Cd$ , since  $A$  is assumed to decide within that time. Since each block takes time  $c_1$ , there must exist a maximal element of  $\Gamma$ , that is, one that has no proper extensions in  $\Gamma$ ; let  $\alpha'$  be such a maximal element.

Let  $\Theta$  be the set of all finite fast ff-extensions of  $\alpha'$  consisting of  $\alpha'$  followed by a single block. In other words, every  $\beta \in \Theta$  consists of  $\alpha'$  followed by a sequence of message deliveries and a single step by each process. Since fast ff-extensions are synchronous,  $t_{\text{end}}(\beta) = t_{\text{end}}(\alpha') + c_1$  for each  $\beta \in \Theta$ . By maximality of  $\alpha'$ , every timed execution prefix in  $\Theta$  is univalent. Since  $\alpha'$  is bivalent, there must be at least one such extension that is 0-valent and at least one that is 1-valent. (This uses the fact that bivalence is by definition with respect to fast ff-extensions.) Let  $\beta'_j \in \Theta$  be  $j$ -valent, for  $j = 0, 1$ .

Now we construct a sequence,  $\beta''_i$ ,  $0 \leq i \leq n$ , of elements of  $\Theta$  such that  $\beta''_0 = \beta'_0$ ,  $\beta''_n = \beta'_1$ , and for all  $i$ ,  $1 \leq i \leq n$ ,  $\beta''_{i-1}$  and  $\beta''_i$  are indistinguishable to all processes other than  $p_i$ . The construction is inductive. First, define  $\beta''_0 = \beta'_0$ . Then, for each  $i$ ,  $1 \leq i \leq n$ , define  $\beta''_i \in \Theta$  to be the same as  $\beta''_{i-1}$  except that the message deliveries to  $p_i$  in  $\beta''_i$  are as in  $\beta'_1$ . (Since all the messages delivered to  $p_i$  in  $\beta'_1$  are sent by time  $t_{\text{end}}(\alpha')$ , such a  $\beta''_i$  exists.)

Since each  $\beta''_i \in \Theta$ , it is univalent. Since  $\beta''_0$  is 0-valent and  $\beta''_n$  is 1-valent, there must exist  $i$ ,  $1 \leq i \leq n$ , such that  $\beta''_{i-1}$  is 0-valent and  $\beta''_i$  is 1-valent. Then defining  $\beta_0 = \beta''_{i-1}$  and  $\beta_1 = \beta''_i$  suffices to prove the lemma.  $\square$

**6.5. THE FINAL STEP.** For the final step of our proof, we now use Lemma 6.2 once again to yield our main lower-bound theorem.

**THEOREM 6.5.** *Assume  $1 \leq f \leq n - 1$ . There is no algorithm in the timing-based model that solves the agreement problem for  $f$  faults within time strictly less than  $(f - 1)d + Cd$ . Moreover, this lower bound holds in the case of synchronized start.*

PROOF. Suppose, by way of contradiction, that such an algorithm  $A$  exists. Then Lemma 6.4 yields an  $(f - 1)$ -admissible synchronous timed execution prefix  $\alpha'$  such that  $t_{\text{end}}(\alpha') \geq (f - 1)d$ ,  $\alpha'$  is bivalent, and there are two fast ff-extensions of  $\alpha'$ , namely,  $\beta_j$  for  $j = 0, 1$ , satisfying the following properties:

- (1)  $\beta_j$  is an extension of  $\alpha'$  by exactly one block,  $j = 0, 1$ ,
- (2)  $\beta_j$  is  $j$ -valent,  $j = 0, 1$ , and
- (3)  $\beta_0$  and  $\beta_1$  are indistinguishable to all but at most one process.

But then  $\beta_0$  and  $\beta_1$  satisfy all the conditions in Lemma 6.2, with  $t = (f - 1)d$ . This immediately yields a contradiction.  $\square$

*Remark 4.* The lower bound obtained in this proof is not always the best possible. If  $d = kc_2 + \epsilon$  for some integer  $k$  and  $\epsilon > 0$ , then we can actually obtain a bound of  $(f - 1)(d + c_2 - \epsilon) + Cd$ . Since in theory  $\epsilon$  can be arbitrarily small, we get essentially  $(f - 1)D + Cd$  in the worst case.

### 7. Implications for Synchronous Processes with Message Delivery Uncertainty

In the Introduction, we indicated that our results could be applied to the model used in [Herzberg and Kuten, 1989], in which process steps are completely synchronous, that is,  $c_1 = c_2$ , so  $C = 1$ , and in which  $\delta$ , the actual message delivery bound in a particular execution, can be much smaller than the worst-case message delivery time  $d$ . In this section, we say more about these applications.

First, we consider the cost of implementing the timeout task in the  $C = 1$  model. The timeout strategy of Section 3 yields a timeout bound  $T$  of at most  $d + \delta + 3c_1$ . However, since processes are synchronous, the timeout bound can be improved slightly, using a different strategy. Process  $p_j$  broadcasts the message  $(alive, j, k)$  at its  $k$ th step for all  $k$ . If process  $p_i$  has not received the message  $(alive, j, k)$  by its  $(k + \lfloor d/c_1 \rfloor + 1)$ -th step, then  $p_i$  adds  $p_j$  to its set of halted processes. This strategy gives a timeout bound of  $T = d + 2c_1$ .

We consider the simple upper and lower bounds for agreement. The simple upper bound of approximately  $(f + 1)Cd$  of Section 4 specializes to yield an upper bound of approximately  $(f + 1)d$ , even for executions in which  $\delta \ll d$ . On the other hand, a simple lower bound, obtained by adapting the  $(f + 1)$ -round lower bound for the rounds model, is  $(f + 1)\delta$ . This leaves a gap of a multiplicative factor of  $d/\delta$ .

The main algorithm of this paper helps to close this gap. Since we carried out the analysis of our main algorithm in terms of  $\delta$  and  $T$ , it is easy to translate the result to the  $C = 1$  model. Using the improved timeout bound above, we conclude that the algorithm runs in time

$$(2f - 1)\Delta + \max\{d, 3\delta\} + 3c_1,$$

or approximately  $(2f - 1)\delta + \max\{d, 3\delta\}$  if  $c_1 \ll \delta$ . Therefore, the number of faults multiplies the actual message delay  $\delta$  rather than the worst-case delay  $d$ .

As shown in [Dwork and Stockmeyer, 1991], the methods of Dwork et al. [1988] give a completely different agreement algorithm in the  $C = 1$  model with time complexity  $O(f\delta)$ , provided that  $n \geq 2f + 1$ . (These methods do not work when  $n \leq 2f$ .)

We now consider lower bounds in the  $C = 1$  model. The lower bound techniques of this paper can be modified to give a lower bound of time  $(2f - n)\delta + d$  provided that  $f + 1 \leq n \leq 2f$  [Dwork and Stockmeyer, 1991]. More specifically, since  $n \leq 2f$ , a “partitioning” argument, similar to ones used in [Bracha and Toueg, 1985] and [Dwork et al., 1988], easily gives a lower bound of  $d$ , even in certain executions in which the actual message delay  $\delta$  is  $c_1$ , so messages are being delivered essentially as fast as possible. By combining the partitioning argument with the argument used to prove the  $(f + 1)$ -round



lower bound (see the discussion preceding Lemma 6.1), a lower bound of  $(2f - n)\delta + d$  can be shown if  $f + 1 \leq n \leq 2f$ . This bound can be compared to the upper bound of roughly  $(2f - 1)\delta + d$  described above. In the case  $n > 2f$ , the upper bound  $O(f\delta)$  shows that the time need not depend on  $d$  at all.

### 8. Conclusions and Open Questions

Although there is a gap between our lower bound of  $(f - 1)d + Cd$  and our upper bound of approximately  $2fd + Cd$ , we feel we have substantially answered the question of how the time requirement depends on the timing uncertainty, as measured by  $C = c_2/c_1$ . In particular, we have shown that only a single “long timeout” (i.e., a timeout requiring time  $Cd$ ) is required and this long timeout cannot be avoided. Similarly, for the case in which  $C = 1$ , we have shown that the time depends on the worst-case message delivery time  $d$  only once.

An obvious open problem is to close the gap between the lower and upper bounds. Another question is whether these results can be extended to other types of failures such as Byzantine or omission failures. Some results on this last question have been obtained by Ponzio [1991a; 1991b].

A more general direction for future research is to try to extend the techniques described in this paper to permit simulation of arbitrary round-based fault-tolerant algorithms in the model with timing uncertainty. The hope is that such a simulation will not incur the multiplicative overhead of  $T$  of the simple transformation described in Section 4.

Our algorithms assume that each message is delivered within at most time  $d$  under all circumstances, in particular, even if the message delivery system is overloaded with messages. A more reasonable assumption is that all messages get delivered within at most time  $d$ , provided that the number of messages in transit is bounded. The algorithms we present in this paper send only a bounded number of messages and so would work under such a restriction. Our lower bound does not rely on this restriction and carries over a fortiori for the restricted case. Some results relating the time complexity of a timeout task to the capacity of the channels appear in [Ponzio, 1991b].

As mentioned earlier, the work presented in this paper is part of an ongoing effort to obtain a precise understanding of the role played by time, and timing uncertainty in particular, in distributed systems. The upper bound presented in this paper is based on an approach that departs from known algorithms for agreement in the synchronous model. We believe that there are many other fundamental tasks in distributed systems whose study might lead to the discovery of new approaches for coping with timing uncertainties.

**ACKNOWLEDGMENTS.** We thank Stephen Ponzio for many helpful discussions and feedback on preliminary versions of this paper, in particular, for first noting the simple lower bound result. We are grateful also to Mark Tuttle for stimulating discussions in the early stages of this research. Michael Merritt helped us realize the relevance of our results to the model introduced by Herzberg and Kutten [1989]. Finally, we thank the two anonymous referees for a careful job and many useful suggestions.

## REFERENCES

- ATTIYA, H., AND LYNCH, N. A. 1989. Time bounds for real-time process control in the presence of timing uncertainty. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*. IEEE, New York, pp. 268–284. Also *Inf. Comput.*, to appear.
- ATTIYA, H., AND MAVRONICOLAS, M. 1990. Efficiency of semi-synchronous versus asynchronous networks. In *Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing*. Univ. Illinois, Urbana-Champaign, Ill. pp. 578–587. Also *Math. Syst. Theory*, to appear.
- BERMAN, P., GARAY, J. A., AND PERRY, K. J. 1989. Towards optimal distributed consensus. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*. IEEE, New York, pp. 410–415.
- BRACHA, G., AND TOUEG, S. 1985. Asynchronous consensus and broadcast protocols. *J. ACM* 32, 4 (Oct.), 824–840.
- COAN, B. A. 1986. A communication-efficient canonical form for fault-tolerant distributed protocols. In *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*. ACM, New York, 1986, pp. 63–72. (A revised version of this paper appears as Chapter 2 of Coan's [1987] Ph.D. dissertation.)
- COAN, B. A. 1987. Achieving consensus in fault-tolerant distributed computer systems: Protocols, lower bounds, and simulation. Ph.D. dissertation. Massachusetts Institute of Technology, Cambridge, Mass.
- COAN, B. A., AND DWORK, C. 1991. Simultaneity is harder than agreement. *Inf. Comput.* 91, 2, 205–231.
- COAN, B. A., AND THOMAS, G. 1990. Agreeing on a leader in real-time. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*. IEEE, New York, pp. 166–172.
- CRISTIAN, F., AGHILI, H., STRONG, H. R., AND DOLEV, D. 1985. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *Proceedings of the 15th International Conference on Fault Tolerant Computing*. 1985, pp. 1–7. Also: IBM Res. Rep. RJ5244 (revised Apr. 1989).
- DEMILLO, R., LYNCH, N. A., AND MERRITT, M. 1982. Cryptographic protocols. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York. 383–400.
- DOLEV, D., DWORK, C., AND STOCKMEYER, L. 1987. On the minimal synchronism needed for distributed consensus. *J. ACM* 34, 1 (Jan.), 77–97.
- DOLEV, D., FISCHER, M. J., FOWLER, R., LYNCH, N. A., AND STRONG, H. R. 1982. Efficient Byzantine agreement without authentication. *Inf. Contr.* 52, 257–274.
- DOLEV, D., HALPERN, J. Y., AND STRONG, H. R. 1986. On the possibility and impossibility of achieving clock synchronization. *J. Comput. Syst. Sci.* 32, 2, 230–250.
- DOLEV, D., REISCHUK, R., AND STRONG, H. R. 1990. Early stopping in Byzantine agreement. *J. ACM* 37, 4 (Oct.), 720–741.
- DOLEV, D., AND STRONG, H. R. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4, 656–666.
- DWORK, C., LYNCH, N., AND STOCKMEYER, L. 1988. Consensus in the presence of partial synchrony. *J. ACM* 35, 2 (Apr.), 288–323.
- DWORK, C., AND MOSES, Y. 1990. Knowledge and common knowledge in Byzantine environments. I: Crash failures. *Inf. Comput.* 88, 2 (Oct.), 156–186.
- DWORK, C., AND STOCKMEYER, L. 1991. Bounds on the time to reach agreement as a function of message delay. IBM Res. Rep. RJ8181. IBM Almaden Research Center, San Jose, Calif.
- FISCHER, M. J., AND LYNCH, N. A. 1982. A lower bound for the time to assure interactive consistency. *Inf. Proc. Lett.* 14, 4 (June), 183–186.
- FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. 1985. Impossibility of distributed consensus with one faulty process. *J. ACM* 32, 2 (Apr.), 374–382.
- HADZILACOS, V. 1984. Issues of fault tolerance in concurrent computations. Ph.D. dissertation. Harvard Univ., Cambridge, Mass. Also Tech. Rep. TR-11-84, Dept. Comput. Sci., Harvard Univ., Cambridge, Mass.
- HALPERN, J. Y., MEGIDDO, N., AND MUNSHI, A. A. 1985. Optimal precision in the presence of uncertainty. *J. Complexity* 1, 170–196.
- HERZBERG, A., AND KUTTEN, S. 1989. Efficient detection of message forwarding faults. In *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing* (Edmonton, Alb., Canada, Aug. 14–16). ACM, New York, pp. 339–353.

- LAMPORT, L., AND FISCHER, M. J. 1982. Byzantine generals and transaction commit protocols. Tech. Rep. Op. 62. SRI International. Menlo Park, Calif.
- LAMPORT, L., AND MELLAR-SMITH, P. 1985. Synchronizing clocks in the presence of faults. *J. ACM* 32, 1 (Jan.), 52–78.
- LAMPORT, L., SHOSTAK, R., AND PEASE, M. 1982. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July), 382–401.
- LUNDELIUS, J., AND LYNCH, N. 1984. An upper and lower bound for clock synchronization. *Inf. Contr.* 62, 2/3 (Aug./Sept.), 190–204.
- MERRITT, M. 1985. Notes on the Dolev–Strong lower bound for Byzantine agreement. Unpublished manuscript.
- MERRITT, M., MODUGNO, F., AND TUTTLE, M. 1991. Time constrained automata. In *Proceedings of CONCUR '91—2nd International Conference on Concurrency Theory* (Amsterdam, The Netherlands, Aug.). Lecture Notes in Computer Science, vol. 527. Springer-Verlag, New York, pp. 408–423.
- MOSES, Y., AND TUTTLE, M. R. 1988. Programming simultaneous actions using common knowledge. *Algorithmica* 3, 121–169.
- MOSES, Y., AND WAARTS, O. 1988. Coordinated traversal:  $(f + 1)$ -round Byzantine agreement in polynomial time. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*. IEEE, New York, pp. 246–255.
- PEASE, M., SHOSTAK, R., AND LAMPORT, L. 1980. Reaching agreement in the presence of faults. *J. ACM* 27, 2 (Apr.), 228–234.
- PONZIO, S. 1991a. Consensus in the presence of timing uncertainty: Omission and Byzantine failures. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, (Montreal, Que., Canada, Aug. 19–21). ACM, New York, pp. 125–138.
- PONZIO, S. 1991b. The real-time cost of timing uncertainty: Consensus and failure detection. S. M. dissertation. Report MIT/LCS/TR-518. Laboratory for Computer Science. MIT, Cambridge, Mass. (Oct.).
- PONZIO, S., AND STRONG, H. R. 1992. Semisynchrony and real time. In *Proceedings of the 6th International Workshop on Distributed Algorithms*. Lecture Notes in Computer Science, vol. 47. Springer-Verlag, New York, pp. 120–135.
- RHEE, I., AND WELCH, J. L. 1992. The impact of time on the session problem. In *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing*. ACM, New York, pp. 191–202.
- SRIKANTH, T. K., AND TOUEG, S. 1987. Optimal clock synchronization. *J. ACM* 34, 3 (July), 626–645.
- STRONG, R., DOLEV, D., AND CRISTIAN, F. 1990. New latency bounds for atomic broadcast. In *Proceedings 11th IEEE Real-Time Systems Symposium*. IEEE, New York, pp. 156–165.
- WELCH, J. L., AND LYNCH, N. 1988. A new fault-tolerant algorithm for clock synchronization. *Inf. Comput.* 77, 1 (Apr.), 1–36.

RECEIVED DECEMBER 1990; REVISED OCTOBER 1992; ACCEPTED OCTOBER 1992