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

(Extended Abstract)

Hagit Attiya† Cynthia Dwork‡ Nancy Lynch§ Larry Stockmeyer‡

Abstract: Upper and lower bounds are proved for the real 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.

Let T denote the worst-case time to detect a failure, i.e., the elapsed time between the failure of some process p and the time when all correct processes determine that p has failed; a straightforward approach yields T roughly equal to Cd. Letting f denote the number of faults to be tolerated, a simple adaptation of an $(f+1)$-round synchronous agreement algorithm takes time $(f+1)T$, or roughly $(f+1)Cd$.

The first principal result of this paper is an agreement algorithm in which the worst-case time T for a timeout is incurred at most *once*, yielding a running time of approximately $2f\delta + T$ in the worst case, where δ is an upper bound on the message delay that actually occurs in a given execution. This represents a significant reduction in complexity in case $C \gg 1$ or $\delta \ll d$. The second principal result is a lower bound of $(f-1)d + Cd$ on the running time of any agreement algorithm, for the case where $\delta = d$; this is close to the upper bound of $2fd + Cd$ for this case.

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

*This work was supported by ONR contract N00014-85-K-0168, by NSF grants CCR-8611442 and CCR-8915206, and by DARPA contracts N00014-89-J-1988 and N00014-87-K-0825.

†Department of Computer Science, Technion. Work performed while the author was at the Laboratory for Computer Science, MIT.

‡IBM Almaden Research Center. Work performed while on sabbatical at the Laboratory for Computer Science, MIT.

§Laboratory for Computer Science, MIT.

the agreement problem we consider, there is a system of n processes, $p_1, \ldots, p_n$, 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 is 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 may send out messages to all processes, receives all messages sent to it at that round, and carries out some local computation. (See, for example, [8, 11, 13, 15, 17, 20, 21, 22, 23, 24] for results involving time complexity in this model.) 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.

We consider how these bounds are affected by using, instead of the synchronous 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.[1] Since the agreement algorithm is not required to be correct if any message delay exceeds d, one might choose a conservative (large) upper bound d. Since actual message delays during particular executions of the algorithm could be smaller than the worst-case bound d, we use a different parameter δ to denote the maximum *actual* message delay. 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.

Let T denote the worst-case time to detect a failure, i.e., the elapsed time between the failure of some process p and the time when all correct processes determine that p has failed; a straightforward implementation of the timeout yields T roughly equal to Cd. To see why d is multiplied by C in the bound, note that in order to ensure that time d has elapsed (e.g., in order to guarantee that no messages remain to be delivered), a process must wait for d/c_1 of its own steps, since it might be running "fast" (i.e., time c_1 between steps). But if the process is actually running "slow" (i.e., time c_2 between steps), the actual waiting time will be $c_2(d/c_1) = Cd$.

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 $(f + 1)T$, or roughly $(f + 1)Cd$. However, the problem turned out to be considerably more complicated. To illustrate this, we consider two interesting cases of the problem and note that, in each case, there is a large gap between the naive upper bound and known lower bounds.

The first case focuses on uncertainty in process speed, i.e., C can take on any value greater than or equal to 1, while assuming that $\delta = d$. Taking $\delta = d$ models a common assumption made in analyzing distributed algorithms, that actual message delays are as large as the worst-case bound d. In this case, as in the general case, the naive upper bound is roughly $(f+1)Cd$. On the other hand, a simple modification to the proof that $f + 1$ rounds are required in the rounds model gives a lower bound of time $(f + 1)d$. This leaves a gap of a factor equal to the timing uncertainty C.

The second case focuses on uncertainty in message delivery time, while assuming that processes

[1] Results of [9, 16] imply that if either one of the bounds 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.

Case	Upper Bound	Lower Bound
$\delta = d$	$2fd + Cd$	$(f-1)d + Cd$
$C = 1$	$(2f-1)\delta + d$	$(2f-n)\delta + d$ if $n \leq 2f$

Table 1: Summary of Time Bounds

take steps in lock-step synchrony at a constant rate. That is, $C = 1$, and the actual message delivery time δ is any value that is less than or equal to the worst-case message delivery time d. (This model is studied in the context of end-to-end communication problems in [19].) In this case, the naive upper bound is roughly $(f+1)d$, even in executions with $\delta \ll d$; thus, the naive algorithm does not take advantage of small message delays when they occur. On the other hand, the $(f+1)$-round lower bound for the rounds model yields a lower bound of time $(f+1)\delta$ in executions with actual message delay δ. This leaves a gap of a factor equal to the ratio d/δ.

Thus, it appears that straightforward extensions of known results for the rounds model do not yield tight bounds for the version of the problem with inexact timing. In this paper, we obtain much closer upper and lower bounds for the timing model. In particular, to a large extent, we answer the question of how the time complexity depends on C and δ.

The first principal result of this paper is an agreement algorithm in which the worst-case time T for a timeout is incurred at most *once*, yielding a running time of approximately $(2f-1)\delta + T$ in the worst case.[2] This represents a significant reduction in complexity over the naive algorithm in case $C \gg 1$ or $\delta \ll d$. An interesting feature of the algorithm is that it can be viewed as an asynchronous algorithm that uses a fault detection (e.g., timeout) mechanism. That is, the timing bounds c_1, c_2, d are used only in the fault detection mechanism. This algorithm uses timing information in a novel way to achieve fast time performance.

We then consider the two cases described above. Our results for these cases are summarized in Table 1. The upper bound in each case is obtained from the time bound of the main algorithm by substituting an upper bound for T. We also prove lower bounds for both cases. The lower bound for the case $\delta = d$ (where the focus is on uncertainty in process step time) is our second principal result. Its proof is unusual in that it combines, in a nontrivial way, three different lower bound techniques: a "chain argument", used previously ([6, 8, 11, 13, 15, 20, 22]) to prove that $f+1$ rounds are required in the synchronous rounds model; a "bivalence" argument, used previously ([9, 16]) to prove that fault-tolerant agreement is impossible in an asynchronous system; and a "time stretching" argument, used previously ([3]) to prove lower bounds for resource allocation problems. Our bounds are reasonably tight: in the $\delta = d$ case, they demonstrate that the time complexity only involves Cd, the timeout bound, in a single additive term; Cd is *not* multiplied by f (the total number of potential failures) as in the naive algorithm. Likewise, in the $C = 1$ case, our results demonstrate that the time complexity only involves d in a single additive term; d is not multiplied by f.

Regarding related work, there has 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 [12]. The main emphasis in [12] 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* (the worst-case elapsed time as measured on the clock of any correct process) of reaching agreement when proces-

[2] We say "approximately" since, in our formal model, a message could be delivered in time δ and it could take an additional time c_2 for the receiver to take a step and send a response. To simplify the expression of our bounds in this Introduction, we approximate the bounds by assuming that $c_2 \ll \delta$ and neglecting the c_2 term. The formal statements of our results in later sections give the precise bounds.

sors are not completely synchronous appears in [5] and [27]. These papers assume that processor 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.

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 (see [26] for a survey). More recently, the mutual exclusion problem has been studied in a timing-based model with $C > 1$ [3]. Also, the time complexity for a synchronizer algorithm to operate in a timing-based network is studied in [4], and the time complexity of leader election algorithms in a timing-based model appears in [7].

2 Definitions

In this section, we outline the definitions for the underlying formal model; complete definitions appear in the full paper [2].

An *algorithm* P consists of n processes $p_1, \ldots, p_n$. Each process p_i is modeled as a (possibly infinite) state machine whose state set contains a distinguished *initial state* and a distinguished *fail* state. A *configuration* C is a vector consisting of local states of all processes. The *initial configuration* is the vector of initial states. 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. Processes can receive *inputs* from some set $\mathcal{V}$ of *values*. Each process p_i follows a deterministic local protocol that determines its state transitions and the messages it sends.

We model a computation of the algorithm as a sequence of configurations alternated with *events*. Events are classified as either process steps, delivery events or input events. A *step* of process p_i is, in turn, either a computation event or a failure event. In a *computation event*, based on its current state, p_i may perform a set S of send actions and may update its state.[3] In a *failure event*, the state determines a set S of send actions as before, but some arbitrary *subset* of these send actions are actually performed. The process p_i then enters the *fail* state. It is convenient to assume that p_i continues to take computation steps after failing, although it stays in the *fail* state and no messages are sent at steps after p_i's failure event.

A *delivery event* at p_i represents the delivery of a message $m \in \mathcal{M}$ to p_i, and an *input event* represents the delivery of an input value $v \in \mathcal{V}$ to p_i.[4] In these events, the process p_i, based on m (or v) and its local state, may change its state.

A *timed event* is a pair (π, t), where π 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 $\alpha = C_0, (\pi_1, t_1), C_1, \ldots, (\pi_j, t_j), C_j, \ldots$, 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 α be a timed sequence as above, we say that α is a *timed execution* of P provided that the following all hold: (a) $C_0, \pi_1, C_1, \ldots, \pi_j, C_j, \ldots$ is a legal execution of P as described above, where C_0 is the initial configuration, (b) each process takes infinitely many steps in α, including a step at time 0, (c) for each p_i, the time between each pair of successive steps of p_i is at least c_1 and at most c_2, and (d) if message m is sent to p_i at time t then the matching delivery of m to p_i occurs at some time t' with $t < t' \leq t + d$.

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

[3] In all our algorithms the set S of send actions will be $broadcast(m)$, that is, $\{send(1, m), \ldots, send(n, m)\}$.

[4] The original definition of the problem in synchronous systems (e.g., [21]) 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 algorithm.

$\Delta = \delta + c_2$.

We now define the *agreement problem*. Let $\mathcal{V}$ be a set of values. We assume that each p_i can irreversibly *decide* on a value $v \in \mathcal{V}$ (say by entering a special decision state). A timed execution α is *f-admissible* if at most f processes fail in α and α contains exactly one input event for each p_i. Define $start(\alpha)$ to be the minimum time t such that, by time t, every p_i has received its input value and taken at least one step.

Fix constants c_1, c_2, d as above and fix a nonnegative integer f. 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 α satisfies the following conditions:

1. (Agreement) No two different processes decide on different values;
2. (Validity) If some process decides on v, then some process receives input value v in α;
3. (Time Bound) Every process either has a failure event or makes a decision by time $start(\alpha) + B(delay(\alpha))$.

We carry out the main development using a Boolean version of the problem, i.e., $\mathcal{V} = \{0, 1\}$. In the full paper ([2]) we show that the main algorithm can be extended to the case of an arbitrary value set.

3 The Algorithm

3.1 The Timeout Task

In the algorithm 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 timeout task of p_i contains the state component *halted*, a subset of $\{1, \ldots, n\}$, containing the indices of processes whose halting (in particular, whose failure) has been detected by p_i. We can describe and analyze the main algorithm without knowing how the timeout task works in detail. We need only that it satisfy the following two properties:

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 $t + T$.

We indicated earlier that a timeout task can be implemented with T roughly equal to Cd; more precisely, the task can be implemented with T approximately equal to $Cd + \delta$. Here we sketch a simple construction. At each step, each process broadcasts a message *alive*. A process stops sending *alive* messages if it decides. 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 has halted.[5] Specifically, "sufficiently many steps" is $\lfloor D/c_1 \rfloor + 1$ steps. Intuitively, a process should wait for about d/c_1 steps to ensure that time d has elapsed. It is not hard to verify that properties T1 and T2 hold with a timeout bound of $T \leq Cd + \delta + (C + 2)c_2$, or approximately $Cd + \delta$ if $c_2 \ll d$.

3.2 Description of the Algorithm

The following is our basic upper bound result.

Theorem 3.1 *Assume the existence of a timeout task with time bound T. There is an algorithm which, for any $f \leq n$, solves the agreement problem for f faults within time*

$$(2f - 1)\Delta + \max\{T, 3\Delta\}.$$

Assuming $c_2 \ll \delta$ and $T \geq 3\delta$, the time is approximately $(2f - 1)\delta + T$. Assuming T is approximately equal to $Cd + \delta$ as above, the time is approximately $2f\delta + Cd$.

Now we give an informal description of the "main" task for process p_i, and we outline the arguments showing that the algorithm is correct. In Section 3.3 we state the key lemmas used in the formal correctness proof. The algorithm is given in more detail in Figure 1 in precondition-effect style. In this code, v_i holds the input value of p_i,

[5] While this strategy gives a good bound in theory, it has high message complexity. A more reasonable approach in practice would be to send the *alive* message only periodically.

Precondition: initial next-phase transition
$r = 0$
$v_i = 1$
Effect:
broadcast$((0, i))$
$r := 1$

Precondition: initial decision transition
$r = 0$
$v_i = 0$
Effect:
broadcast$((1, i))$
decide(0)

Precondition: next-phase transition
$r \geq 1$
there exists a j such that $(r, j) \in buff$
Effect:
broadcast$((r, i))$
$r := r + 1$

Precondition: decision transition
$r \geq 1$
for all $j \notin halted$, $(r-1, j) \in buff$
there is no j such that $(r, j) \in buff$
Effect:
broadcast$((r+1, i))$
decide$(r \bmod 2)$

Figure 1: The "main" task of the agreement algorithm for process p_i.

i.e., an input event which delivers v to p_i sets v_i to v and causes the algorithm to start. The local variable r is a *phase number*, initially 0. The state component *buff* is a message buffer which holds all received 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 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, in this case the algorithm ensures that every non-failed, 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 non-failed 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 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 does not decide at 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 non-halted 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, 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 non-failed process broadcasts $r+1$ and goes on to phase $r+2$. Then every undecided, non-failed process will receive the message $r+1$ from all non-halted processes, and no message $r+2$ from any process. It follows that each undecided, non-failed 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 non-failed 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, let r be a phase during which no process fails; such a phase must occur among

phases $0, 1, \ldots, f$, so $r \leq f$. If some process decides at a phase numbered at most r, then as argued above all non-failed processes decide by phase $r + 2 \leq f + 2$. On the other hand, if no process has decided up to and including phase r, then no process sends the message $r + 1$ and all non-failed processes broadcast the message r. So the preconditions for every process to decide at phase $r + 1$ are satisfied.

Remark. 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. [10]). As noted above, the algorithm also has the property that if all initial values are the same then all nonfaulty processes decide by the end of phase 1, regardless of the number of faults.

3.3 Correctness Proof

We first give a definition that is 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 the message (r, j) to p_i.

We now state the key lemmas used in the proof of correctness. The proofs are similar to the arguments outlined above, and all details are given in [2]. We give here the proofs of Lemmas 3.4 and 3.5 since they are short and illustrate why a quiet phase is a useful concept and how it is used. When we say that a process *begins* a transition, we mean that the precondition for the transition is satisfied and either the associated computation event or an associated failure event is performed. The first lemma shows that nonfaulty processes do not get "stuck" in a phase.

Lemma 3.2 *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 next-phase or decision transition at phase r.*

Lemma 3.3 *If some process decides at phase $r \geq 0$, then no process begins a decision transition at phase $r + 1$ (so no process decides at phase $r + 1$).*

Lemma 3.4 *If phase r is quiet, then all processes either fail or decide by the end of phase r.*

Proof: If some p_j does not fail or decide by the end of phase r, then by Lemma 3.2 it successfully broadcasts the message (r, j) while executing a next-phase transition at phase r. This contradicts the assumption that phase r is quiet. ■

Lemma 3.5 *Assume that some process decides at phase r. Then phase $r+2$ is quiet (so all processes either fail or decide no later than phase $r + 2$).*

Proof: By Lemma 3.3, no process begins a decision transition at phase $r + 1$. Since the earliest sending of a message containing $r + 2$ must occur at a decision transition at phase $r + 1$, it follows that phase $r + 2$ is quiet. ■

Since a process must decide $r \bmod 2$ at phase r, Lemmas 3.3, 3.4, and 3.5 imply the agreement property.

Termination follows from Lemma 3.4 and the following:

Lemma 3.6 *Any f-admissible timed execution contains a quiet phase, numbered no larger than $f + 2$.*

The validity property is easy to prove, as outlined above.

3.4 Timing Analysis

We outline the proof that 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. (The complete proof appears in the full paper [2].)

Fix an arbitrary f-admissible timed execution α. All definitions are with respect to α.

Note that the only transition that occurs because of a timeout is the (non-initial) decision transition. Suppose this transition is begun by a process p_i at a phase h and no (h, j) message *ever* arrives at p_i; in particular, phase h is quiet. Then the timeout can take time T, but then by Lemma 3.4 all nonfaulty processes will decide no later than phase h.

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 decision transition (or the transition would not be enabled). For each $r \geq 1$, denote by f_r the number of processes whose failure step is a transition during which the message r should be broadcast. Then we claim that each such phase r takes only time depending on $f_r\delta$, but not on T. 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 $f_r + 1$ (because a non-failed 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. Note that a process has at most one failure step and thus, in all f-admissible executions, $\sum_{r\geq 1} f_r \leq f$.

More precisely, for $r \geq 0$, define t_r to be the minimum time t such that all processes either fail, decide, or perform a transition from phase r to phase $r+1$ no later than time t. Note that $t_r \leq t_{r+1}$ for all r, and $t_0 \leq s$ where $s = start(\alpha)$. We show:

Lemma 3.7 *For any non-quiet phase $r \geq 1$, $t_r \leq t_{r-1} + \Delta(f_r + 1)$.*

Let h be the smallest number of a quiet phase. By induction we have:

Corollary 3.8 *For every r with $1 \leq r \leq h-1$, $t_r \leq \Delta \cdot \sum_{i=1}^{r}(f_i + 1) + s$.*

To summarize, Corollary 3.8 bounds the time taken up to the first quiet phase h. Phase h can take time T, but all non-failed processes decide no later than phase h. Recall also from Lemma 3.6 that $h \leq f + 2$.

We now prove an upper bound result that is slightly weaker than the one in Theorem 3.1.

Theorem 3.9 *There is an algorithm to solve the agreement problem for f faults within time $(2f + 1)\Delta + T$.*

Proof: By Lemma 3.4, all processes either fail or decide no later than time t_h. It is easy to see, for any phase r, that $t_r \leq t_{r-1} + T$. Therefore, all processes either fail or decide no later than time $t_{h-1} + T$. Now

$$\begin{aligned} t_{h-1} + T &\leq \Delta \cdot \textstyle\sum_{i=1}^{h-1}(f_i + 1) + T + s \\ &\quad \text{by Corollary 3.8,} \\ &\leq (f + (h-1))\Delta + T + s \\ &\leq (2f + 1)\Delta + T + s \\ &\quad \text{by Lemma 3.6 .} \end{aligned}$$

■

The smaller bound given in Theorem 3.1 requires a finer analysis which we leave to the full paper [2]. For $C \geq 2$, there is an example showing that the smaller bound is close (within $O(c_2(C + f))$) to the actual worst-case running time of the algorithm. The better bound is obtained by considering the latest time at which a failure occurs. The time T taken by the timeout task can then be measured starting from the time of the latest failure.

4 Upper and Lower Bounds for Two Cases

In this section, we consider the two cases described in the Introduction, emphasizing the uncertainty in process step time and message delivery time, respectively. In each case, we specialize our algorithm to obtain an upper bound, and we separately prove a corresponding lower bound result.

4.1 The Case $\delta = d$

In this case, the general upper bound specializes to yield approximately $2fd + Cd$. To simplify the expression of the exact bound, we assume $C \geq 2$.[6] Recall that $D = d + c_2$.

Theorem 4.1 *Assume $C \geq 2$. There is an algorithm which, for any $f \leq n$, solves the agreement problem for f faults within time $2fD + CD + c_2$.*

[6] If $C < 2$, the straightforward upper bound $(f + 1)Cd$ is smaller than $2fd + Cd$.

We can prove a lower bound which has a similar form to the upper bound, except that $2fd$ is replaced by $(f-1)d$. The proof of the lower bound $(f-1)d+Cd$ requires three steps and employs techniques used elsewhere in proving lower bounds and impossibility results in the synchronous 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 [6, 8, 11, 13, 15, 20, 22]. This yields the existence of two "long" execution prefixes (taking time at least $(f-1)d$), each having only $f-1$ faults, distinguishable only to one correct process, and each extendible to an execution with a different decision value. The second step mimics a key lemma in the proof that agreement is impossible in asynchronous systems [9, 16]. In this step it is shown that at least one of the two long execution prefixes just described is actually "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. This long bivalent execution is developed a bit more in a technical lemma that shows it can be extended to a "maximal" fast bivalent execution containing at most $f-1$ faults. The last step exploits the one remaining fault, via a technique of [3], to show that after this maximal bivalent execution at least one "long timeout" (taking time at least Cd) is necessary. Leaving the many details to the full paper [2], we state the lower bound result.

Theorem 4.2 *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 $(f-1)d+Cd$. This lower bound holds even in the case that all processes receive their input values at time 0.*

4.2 The Case $C = 1$

In the case where $C = 1$, we can use an "optimized" timeout task that works with $T \leq d+2c_1$. Using Theorem 3.1 again, we obtain an upper bound of approximately $(2f-1)\delta+d$. To simplify the expression of the exact bound, we assume $d \geq 3\delta$.

Theorem 4.3 *Assume $C = 1$ and $d \geq 3\delta$. There is an algorithm which, for any $f \leq n$, solves the agreement problem for f faults within time $(2f-1)\Delta + d + 3c_1$.*

By adapting part of the argument used in the previous lower bound, and adding one new idea, we show the following lower bound; the proof is given in [14].

Theorem 4.4 *Assume $C = 1$ and $f+1 \leq n \leq 2f$. There is no algorithm in the timing-based model that solves the agreement problem for f faults within time strictly less than $(2f-n)\delta+d$. This lower bound holds even in the case that all processes receive their input values at time 0.*

Note that if $n = f+1$, the lower bound is $(f-1)\delta+d$ which is similar to the previous lower bound. As n approaches $2f$, however, the lower bound degenerates to d. Concerning the case $n > 2f$, methods of [12] give an agreement algorithm with running time $O(f\delta)$, showing that the time bound need not depend on d at all in this case; the details can be found in [14].

5 Conclusions and Open Questions

Although there is a gap between our lower bound of $(f-1)d+Cd$ and our upper bound of $2fd+Cd$, 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. We reach a similar conclusion for the $\delta < d$ case.

An obvious open problem is to close the gaps that remain between the upper and lower bounds for the two cases. 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 [25].

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 the Introduction. If a problem can be solved by reduction to agreement (by first agreeing on the input values of all processes and then locally applying some function to the agreed upon values), then our algorithm can be used to solve the problem within time approximately $2f\delta + T$. However, there are problems whose round complexity is significantly smaller than the round complexity of agreement; for example, an algorithm in [18] can be used to solve the *renaming* problem ([1]) within $O(\log n)$ rounds, even in the presence of up to $n-1$ faults. For such problems, reducing the dependency on T is achieved at the cost of increasing the dependency on δ.

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.

Acknowledgements:
We thank Stephen Ponzio for many helpful discussions and feedback on preliminary versions of this paper. 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 Theorem 3.1 to the model of [19].

References

[1] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg and R. Reischuk, "Renaming in an Asynchronous Environment," *JACM,* Vol. 37, No. 3 (July 1990), pp. 524–548.

[2] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer, "Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty," Report TM-435, Laboratory for Computer Science, MIT. Also: IBM Research Report RJ7853.

[3] H. Attiya and N. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," *Proc. 10th IEEE Real-Time Systems Symposium,* 1989, pp. 268–284. Also: Technical Memo MIT/LCS/TM-403, Laboratory for Computer Science, MIT, July 1989.

[4] H. Attiya and M. Mavronicolas, "Efficiency of Asynchronous vs. Semi-Synchronous Networks," *28th Allerton Conf. on Communication, Control and Computing,* October 1990. Also: Technical Report 21-90, Department of Computer Science, Harvard University, September 1990.

[5] F. Cristian, H. Aghili, R. Strong and D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," *Proc. 15th Int. Conf. on Fault Tolerant Computing,* 1985, pp. 1–7. Also: IBM Research Report RJ5244, revised October 1989.

[6] B. A. Coan and C. Dwork, "Simultaneity is Harder than Agreement," *Proc. 5th IEEE Symp. on Reliability in Distributed Software and Database Systems,* 1986, pp. 141–150.

[7] B. A. Coan and G. Thomas, "Agreeing on a Leader in Real-Time," *Proc. 11th IEEE Real-Time Systems Symposium,* 1990.

[8] R. DeMillo, N. Lynch and M. Merritt, "Cryptographic Protocols," *Proc. 14th Annual ACM Symp. on Theory of Computing,* May 1982, pp. 383–400.

[9] D. Dolev, C. Dwork and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *JACM,* Vol. 34, No. 1 (January 1987), pp. 77–97.

[10] D. Dolev, R. Reischuk, and H. R. Strong, "Early Stopping in Byzantine Agreement,"

JACM, Vol. 37, No. 4 (Oct. 1990), pp. 720–741.

[11] D. Dolev and H. R. Strong, "Authenticated Algorithms for Byzantine Agreement," *SIAM J. on Computing*, Vol. 12, No. 4 (November 1983), pp. 656–666.

[12] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *JACM*, Vol. 35 (1988), pp. 288–323.

[13] C. Dwork and Y. Moses, "Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures," *Information and Computation*, Vol. 88, No. 2 (1990), pp. 156–186.

[14] C. Dwork and L. Stockmeyer, "Bounds on the Time to Reach Agreement as a Function of Message Delay," IBM Research Report, in preparation.

[15] M. Fischer and N. Lynch, "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, Vol. 14, No. 4 (June 1982), pp. 183–186.

[16] M. Fischer, N. Lynch and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *JACM*, Vol. 32, No. 2 (1985), pp. 374–382.

[17] V. Hadzilacos, *Issues of Fault Tolerance in Concurrent Computations*, Ph.D. Thesis, Harvard University, June 1984. Technical Report TR–11–84, Department of Computer Science, Harvard University.

[18] M. Herlihy and M. Tuttle, "Wait-Free Computation in Message-Passing Systems," *Proc. 9th ACM Symp. on Principles of Distributed Computing*, 1990, pp. 347–362.

[19] A. Herzberg and S. Kutten, "Efficient Detection of Message Forwarding Faults," *Proc. 8th ACM Symp. on Principles of Distributed Computing*, 1989, pp. 339–353.

[20] L. Lamport and M. Fischer, "Byzantine Generals and Transaction Commit Protocols," Tech. Report Op. 62, SRI International, Menlo Park, CA, 1982.

[21] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Trans. on Prog. Lang. and Sys.*, Vol. 4, No. 3 (July 1982), pp. 382–401.

[22] M. Merritt, "Notes on the Dolev-Strong Lower Bound for Byzantine Agreement," unpublished manuscript, 1985.

[23] Y. Moses and M. Tuttle, "Programming Simultaneous Actions Using Common Knowledge," *Algorithmica*, Vol. 3 (1988), pp. 121–169.

[24] M. Pease, R. Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults," *JACM*, Vol. 27, No. 2 (1980), pp. 228–234.

[25] S. Ponzio, "Real-time Analysis of Timing-based Distributed Algorithms," MS thesis, in progress, MIT Electrical Engineering and Computer Science, 1990.

[26] B. Simons, J. L. Welch and N. Lynch, "An Overview of Clock Synchronization," *Proceedings of IBM Fault-Tolerant Computing Workshop*, March, 1986.

[27] R. Strong, D. Dolev and F. Cristian, "New Latency Bounds for Atomic Broadcast," *Proc. 11th IEEE Real-Time Systems Symposium*, 1990.