New Latency Bounds for Atomic Broadcast

(EXTENDED ABSTRACT)

Ray Strong Danny Dolev Flaviu Cristian
IBM Research Division
Almaden Research Center

Abstract

This paper provides new tighter bounds on the time required to reach agreement in a distributed system as a function of the failure model.

1 Introduction

The time required to reach agreement has been the subject of a number of studies in the literature of fault tolerant distributed systems (see, for example, [DM, DS, DRS, FL, H, NT, PSL]). For many communication and failure models, previous results have completely characterized this time. However, most of these results correspond to models in which processors are perfectly synchronized, either communicating in synchronized rounds or using clocks that present an exact Newtonian time frame (real time). In this paper we study a model that is more sensitive to timing problems because clocks are assumed to be only approximately synchronized. Here much less is known about the time required to reach agreement. In this paper we study agreement as the result of a single execution of an atomic broadcast protocol. An input message is given to one processor in a system and, if everything works correctly, the message is delivered as output by all processors in the system. Relative to a given failure model, an atomic broadcast protocol is said to *tolerate* the failure of r components in a network, if, in every execution of the protocol in the network in which at most r components fail, in each connected subnetwork of correct components, all processors deliver the same message (if any), and, if there is such a connected subnetwork that contains the processor that was given the input message, then all its processors deliver the input message as output. In order to capture formally the notion of "time required to reach agreement," we define the *latency* of an atomic broadcast algorithm as the worst case difference between the clock time of delivery of an atomic broadcast message (by a correct processor) and the timestamp on the message (corresponding to its clock time of origin if the originating processor is correct).

In one of the earliest studies of our model, a 1985 paper on atomic broadcast [CASD], the authors provide a sequence of algorithms for atomic broadcast in distributed systems. The first of these algorithms tolerates omission failures; the second, timing failures; and the third, authentication-detectable Byzantine failures. The latency of the [CASD] omission failure tolerant algorithm is known to be optimal for many networks, including the completely connected network; and it has long been conjectured that the latencies of the [CASD] timing and authentication-detectable Byzantine tolerant algorithms were also optimal on these networks. But a proof of optimal latency was

only available for the third algorithm and only for the case of a network of n processors with $n-2$ failures to be tolerated. In this paper we disprove both of the conjectures by providing a new timing failure tolerant algorithm and a new authentication-detectable Byzantine failure tolerant algorithm. We show that the best possible latency for timing failure tolerance is very different from the best possible latency for authentication-detectable Byzantine failure tolerance.

We had previously shown that omission failures require an atomic broadcast latency of the form $D+e$, where D depends only on the network, network transmission delays, and the number of failures to be tolerated (independent of clock synchronization precision) and e is the maximum deviation possible between correct clocks (independent of the number of failures to be tolerated). In contrast, it was shown that clock failures can require a latency of the form $D+(n-1)e$ when the number of failures to be tolerated is $n-2$. By constructing a new timing failure tolerant algorithm, we now have the surprising result that the latency for timing failures can be expressed in the form $D+e$, implying that algorithms tolerant of timing failures cannot always be converted to algorithms tolerant of clock failures without a performance time penalty. Thus we have shown a fundamental difference between clock and timing failures. We also provide an authentication-detectable Byzantine failure tolerant algorithm with a latency that can be expressed in the form $D+(\lfloor\frac{n}{n-f}\rfloor+\lfloor\frac{n-1}{n-f}\rfloor)e$, where f is the number of faulty processors to be tolerated. We show that this latency is optimal with respect to the coefficient of e for clock and omission failure tolerant algorithms. Using the conversion technique of Srikanth and Toueg, we convert this algorithm to an arbitrary failure tolerant algorithm with a latency that can be expressed in the form $D+2e$, provided $f<\frac{n}{3}$. These algorithms and lower bounds are the main technical results of our paper. They are found in Section 4.

We were led to these algorithms from an investigation of the optimality proof and an attempt to characterize a minimal class of failures for which it held. As a result we can now describe a much more general class of failures that is tolerated by the second algorithm of [CASD]. The failure classes we study in this paper were formulated to extend our understanding of atomic broadcast and related algorithms and to allow more precise comparisons of their failure tolerance.

After describing the model of a distributed system that is a context for this work in Section 2, we define several failure classes in Section 3. In Section 4, we define a partial order on classes of failures that involves whether there is a latency penalty in converting from tolerance of one failure class to another. In this setting we distinguish clock and timing failures, showing that there can be a penalty in converting from timing failure tolerance to clock failure tolerance. We leave open the exact expression for the optimal latency for timing failure tolerant atomic broadcast, though we conjecture that there is some penalty in converting from omission failure tolerance to timing failure tolerance.

2 The Model

We assume a distributed system consisting of n processors completely connected by links. Space limitations preclude a more formal description of our model. We assume that, in any finite interval of real time, only finitely many events take place in the entire system.

For each component we assume there is a correctness *specification* consisting of a relation between finite sequences of external events and single output events. A more formal discussion of specifications will be given

in the complete paper.

We denote the reading of the clock of processor q at real time t by $C_q(t)$. Clocks are specified to maintain linear envelope synchronization: for all real times $u < v$ and for all processors p and q,

$$a_1(v-u) < C_q(v) - C_q(u) < a_2(v-u) + a_3$$

and

$$|C_p(v) - C_q(v)| \leq e$$

for positive constants a_1, a_2, a_3, and e (cf. [DHSS]). Note that a consequence of the linear envelope synchronization is that any interval timed to be of length x by one correct clock cannot be timed to be of length more than

$$drift(x) = (a_2/a_1)(x) + a_3$$

on any other correct clock.

Links are specified to transmit messages from processor to processor within a bounded amount of real time. Processors are specified to respond to messages according to the given protocol within a bounded amount of real time, the response including the output of any messages required by the protocol. We assume processing time is negligible. We simplify and summarize these bounds with a constant d such that, if u is the real time at which processor p outputs message m on link l, v is the real time at which processor q, at the other end of link l, finishes processing m and has output all messages required by the protocol in response to m, and r is any processor in the system, then

$$0 < C_r(v) - C_r(u) \leq d.$$

In the proofs of Theorem 4.5 and Theorem 4.7 we will use a bound ρ on the relative rate of drift defined as follows:

$$\rho = \frac{a_2}{a_1} + \frac{a_3}{2d} - 1.$$

This ρ has the property that, whenever $x \geq 2d$, we have $drift(x) \leq (1+\rho)x$. We will also use an accumulated form of ρ as follows:

$$\gamma(j) = \sum_{i=0}^{j} (1+\rho)^i.$$

Note that if $\rho = 0$ then $\gamma(j) = j + 1$.

3 Failure Classes

We provide a failure classification in order to compare the fault tolerance of algorithms. For this reason, we restrict our discussion to the classification of sequences of external events. A *failure* is a local history of a component that does not satisfy the specification of the component. Because of our bounded time span assumption, any failure has a finite prefix that does not satisfy the specification. The *failure event* of a failure is the first event e in the local history that makes it a failure, that is, the local prefix ending in e does not satisfy the specification, but the local prefix ending in the prior event does. The *failure time* of a failure is the real time of its failure event.

If the addition of output events to the local history would satisfy the correctness specification, then the failure is classified as an *omission failure*. If changing the real time of some output events would satisfy the correctness specification, then the failure is classified as a *timing failure* [CASD].

Components communicate in the distributed system by the identification of certain types of output action at one component with types of input action at another component. If the removal of input events from a failure would make it satisfy the specification, then the failure is classed as a *pseudo input omission failure*. The term "pseudo" is used here to emphasize that the output behavior of the component is *as if* some input event had not occurred. Notice that a

pseudo input omission failure is not necessarily an omission failure. The class containing omission and pseudo input omission failures is called the class of *generalized omission failures* [NT]. (In [CASD] pseudo input omission failures were not included in any class except the class of (universal) Byzantine failures.) When the component behaves as if certain input events occurred either before or after the output events with which they are identified, then the failure is classified as a *pseudo input timing failure*. The class containing timing and pseudo input timing failures is called the class of *generalized timing failures*.

Relative to a fixed authentication protocol (v. [LSP, DS, CASD]), the class of authentication-detectable Byzantine failures is defined as the class of arbitrary failures that do not corrupt the authentication protocol, so that a correct processor can send a copy of an authenticated message to any other processor, proving that the supposed originator of the authenticated message did in fact send the message.

We focus on one special type of subcomponent of a processor called a *clock*. Clocks are assumed to have an input "read" event. In response to a "read" event a clock is specified to deliver a reading (integer) to the subcomponent that issued the "read" request. A clock can only fail by omitting to issue a reading in response to a "read" event or by issuing a reading value that violates the given linear envelope requirement of section 2. When the failure of a processor could be explained by a failure of its clock, then the failure is classed as a *clock failure*. For simplicity we will restrict attention to clock failures that cannot be detected by considering only local history, independent of the actual passage of real time. (Assume that there is a failure free component that watches the input-output behavior of the clock and stops the processor if it can detect a clock failure without reference to any external source of real time. This discussion will be expanded in the complete paper.)

4 New Latency Bounds for Completely Connected Networks

In this section we assume that the network of n processors is completely connected.

Hadzilacos [H] and Neiger and Toueg [NT] have studied the possibility of converting algorithms tolerant of one class of failures into algorithms tolerant of a more extensive class of failures. The object of such a conversion is to minimize any conversion overhead cost and provide a useful but more fault tolerant algorithm. In particular, Hadzilacos was the first to observe that crash failure tolerant atomic broadcast protocols could be converted to omission failure tolerant atomic broadcast protocols with no added latency.

Relative to this environment, we say class C of failures *dominates* class D if, for every atomic broadcast algorithm that tolerates failures from class D at any f of the n processors, there is an atomic broadcast algorithm with the same or shorter termination time that tolerates failures from class C at any f of the n processors. Note that e is implicitly universally quantified in this definition, so the "dominance" must hold for any ratio between d and e.

In this environment, relative to a given number n of processors and a given maximum number of faulty processors f, we say a class C of failures has *precision sensitivity* k if some atomic broadcast algorithm tolerant of class C has a latency of $ke + O(d)$ and any atomic broadcast algorithm tolerant of class C must have a latency of at least ke, no matter how large the ratio between e and d. To show that class C does not dominate class D it suffices to produce n and f for which the

precision sensitivity of class D is larger than that of class C.

In [CASD] it was shown that the class of omission failures has precision sensitivity 1. It was also shown that the class of authentication-detectable Byzantine failures has precision sensitivity $n-1$ when $f = n-2$. It has long been conjectured that the atomic broadcast algorithms of [CASD] have optimal latency for the failure classes they tolerate. If this were true then the precision sensitivity for any failure class that includes timing failures would be at least $f + 1$. We will disprove this conjecture by showing that, when the network of n processors is completely connected, then the class of clock and generalized omission failures has a precision sensitivity of exactly $\lfloor \frac{n}{c} \rfloor + \lfloor \frac{n-1}{c} \rfloor$, where $n = f+c$, and that the class of timing failures has a precision sensitivity of 1.

We first prove some combinatorial prerequisites. We define $k(n,r) = \lfloor \frac{n}{r} \rfloor + \lfloor \frac{n-1}{r} \rfloor$. For the following lemmas we assume that $0 < f < n-1$ so that $1 < c < n$.

Lemma 4.1: If $\{S_i | 1 \leq i \leq j\}$ is a family of disjoint nonempty subsets of a set of cardinality n and if, for every i with $1 \leq i < j$, we have $|S_i| + |S_{i+1}| \geq c$, then $j \leq k(n,c) + 1$. Moreover, for any n and f, there is such a family with $j = k(n,c)+1$ and with $|S_i|+|S_{i+1}| = c$, for every i with $1 \leq i < j$.

Proof Sketch: Let $x = \lfloor \frac{n}{c} \rfloor$ so that $n = (c)x + y$ with $0 \leq y < c$.

Suppose there is a family of subsets satisfying the hypotheses. Let $a = |S_1|$. Without loss of generality $1 \leq a < c$. Let $b = |S_2|$. Again without loss of generality, $b = c - a$. Moreover, without loss of generality, the cardinalities of the subsets alternate between a and b.

Let z be the number of subsets of cardinality b. There are at least as many subsets of cardinality a. Since $z(a + b) \leq n$, it must be the case that $z \leq x$. If there are $z+1$ subsets of cardinality a, then $z(a + b) + a \leq n$, so either $z < x$ or $a \leq y$. Thus either (1) $y = 0$ and $j = 2z \leq 2x$ or (2) $y > 0$ and $j \leq 2z + 1 \leq 2x + 1$.

If $y = 0$, then $\lfloor \frac{n-1}{c} \rfloor = x - 1$, so $k(n,c) = 2x - 1$. If $y > 0$, then $\lfloor \frac{n-1}{c} \rfloor = x$, so $k(n,c) = 2x$. Thus in either case $j \leq k(n,c) + 1$.

Now let n and f be arbitrary, subject to $0 < f < n-1$ and $n = f+c$. If $y = 0$ then let $a = 1$ and let $b = c - a$. Then the set with n elements can easily be partitioned into x sets of size a and x sets of size b. If $y > 0$ then let $a = y$ and let $b = c - a$. Then the set with n elements can be partitioned into $x + 1$ sets of size a and x sets of size b. In either case a family of subsets of size $k(n,c) + 1$ satisfies the hypotheses. □

Lemma 4.2: If $\{R_i | 1 \leq i \leq j\}$ is a family of subsets of a set of cardinality n, if, for every i with $1 \leq i \leq j$, we have $|R_i| \geq c$, and if each subset R_i has nonempty intersection only with R_{i-1} and R_{i+1}, then $j \leq k(n,c)$.

Proof:

For each i with $1 \leq i \leq j-1$, let $S_i = R_i - R_{i+1}$. Let $S_j = R_j \cap R_{j-1}$ and let $S_{j+1} = R_j - R_{j-1}$. The family of subsets $\{S_i | 1 \leq i \leq j+1\}$ satisfies the hypotheses of Lemma 4.1, so $j + 1 \leq k(n,c) + 1$. □

Theorem 4.3: If $0 < f < n - 1$ and $n = f + c$, then $k(n,c)(d + e)$ is a lower bound on the latency of any atomic broadcast protocol that tolerates clock and generalized omission failures in any f processors.

Proof Sketch:

Let $k = k(n,c)$. By Lemma 4.1 we can partition the set of processors into $k+1$ disjoint

sets $\{S_i | 0 \leq i \leq k\}$ such that the union of any pair of adjacent sets has cardinality exactly c.

Suppose there were an atomic broadcast protocol with a latency less than $k(d+e)$. Consider only executions in which, at real time t the clocks in S_i all read $t+ie$, and each message transmission and processing that completes requires exactly d units of real time. (Note that such executions are consistent with our model assumptions from section 2.) In any such execution there are exactly f faulty processors and the correct processors reside in two adjacent sets S_i, S_{i+1}. Thus, using generalized omission failures, we can produce executions of this type in which any message called for by the protocol between processors that are not in adjacent sets is either omitted by the sending processor or the receiving processor acts as if it did not receive the message. Consider an execution of this type in which communication is constrained in this way and otherwise all processors obey the protocol. If the correct processors reside in S_{k-1} and S_k, then the message delivered as output by correct processors must be some default or empty message, because information about the input message does not have time to reach the processors of S_k before the latency is exhausted. Using the idea that correct processors in S_i cannot tell which of S_{i-1} and S_{i+1} are also correct, it is now straightforward to prove by induction that, even when the correct processors are in S_0 and S_1, the correct processors must give an empty or default message as output; but this contradicts the specification of the protocol. □

Corollary 4.4: The precision sensitivity of any class of failures containing both clock and generalized omission failures is at least $k(n,c)$. Thus the class of omission failures does not dominate the class of clock and generalized omission failures.

Note that when $0 < f < \frac{n}{2}$, then $k(n,c) = 2$.

Theorem 4.5: For $0 < f < n-1$ with $n = f + c$, there is an atomic broadcast protocol for n completely connected processors that tolerates authentication detectable Byzantine failures in up to f processors and has a latency of $k(n,c)e + \alpha d$, where α depends only on n, f, and ρ.

Corollary 4.6: The precision sensitivity of the class of clock and generalized omission failures is exactly $k(n, n-f)$.

Proof Sketch of Theorem 4.5:

First we give pseudocode for the atomic broadcast protocol. We assume a *send* operation that timestamps, signs with an authentication protocol, and sends a message to all neighbors. We also assume a *receive* operation that checks all signatures with the authentication protocol, checks to see that each message timestamped t is received after $t-e$ and by $t+e+d$, and checks to see that any imbedded timestamps and signatures in the message structure (defined below) correspond to messages that would have passed the above tests.

Variable	**: type**
z	: value
m	: message
p, q, r, x	: processor
P, Q, R	: set of p
R'	: set of p, q
s, y	: signature
t, u, v	: time
S	: set of p, s, t, u
i, j	: integer

Message formats:

amessage := adopt(z) |
adopt(amessage,p, S)
emessage := echo(amessage,p, s, t, u)
rmessage := reecho(z, p, s, t)
message := amessage |
emessage | rmessage

Procedure timetest(m, p, s, t, u, v) :
$j \leftarrow depth(m)$
for $i = 1$ to j do
$h \leftarrow h(base(i, m, p, s, t))$
if $u > h + (j - i - 1)d + 2d\gamma(j - i))$
then discard message
and end processing fi od
if $g(m, p) = \perp$ then
$g(m, p) \leftarrow min(u, t + e)$ fi
if $u > g(m, p) + v$ then
discard message
and end processing fi
end.

Procedure addechoes(m, q, p, s, t, u) :
if $p \notin R(m, q)$ then
add p to $R(m, q)$
add p, s, t, u to $S(m, q)$
if $|R(m, q)| \geq c\&q \in R(m, q)$ then
unless already sent
send adopt($m, q, S(m, q)$) fi fi
end.

Procedure depth(m) :
if m is of form adopt(z)
then return 1
else if m is of form adopt(m_1, p_1, S)
then return $1 + depth(m_1)$
fi fi
end.

Procedure base(j, m, p, s, t) :
if $depth(m) \leq j$ then return m, p, s, t
else if m is of form adopt(m_1, p_1, S)
then find element $p_1, s_1, t_1, u_1 \in S$
return $base(j, m_1, p_1, s_1, t_1)$
fi fi
end.

Procedure timeset(m, p, s, t, u) :
if $h(m, p, s, t) = \perp$ then
$h(m, p, s, t) \leftarrow u$
send $reecho(m, p, s, t)$ fi
end.

Procedure processmessage:
if processor x receives amessage m
at local clock time u directly
from processor p with signature s
and timestamp t then
if $depth(m) = f + 1$ then
unless already done
deliver $(base(1, m, p, s, t))$
else if $depth(m) < f + 1$ then
for $i = 1$ to $depth(m)$ do
$timeset(base(i, m, p, s, t), u)$
od
timetest(m, p, s, t, u, d)
send echo(m, p, s, t, u)
with signature y
and timestamp u
addechoes(m, p, p, s, t, u)
addechoes(m, p, x, y, u, u)
fi fi fi
if processor x receives
emessage echo(m, p, s_1, t_1, u_1)
at local clock time u
directly from processor q
with signature s
and timestamp t
and depth(m) $< f + 1$ then
for $i = 1$ to $depth(m)$ do
$timeset(base(i, m, p, s_1, t_1), u)$ od
timetest($m, p, s_1, t_1, u, 2d$)
addechoes(m, p, q, s, t, u) fi
if processor x receives
rmessage reecho(m, p, s, t)
at local clock time u then
for $i = 1$ to $depth(m)$ do
$timeset(base(i, m, p, s, t), u)$ od fi
end.

Procedure initiate(z) :
send $adopt(z)$ with signature y

(from processor x) and timestamp t
deliver(*adopt*(z), x, y, t)
end.

Discussion:

Consider a received amessage with its nested sets of timestamps. Divide the interval of time over which the timestamps spread (the interval of latency) into subintervals as follows: Let t_1 be the latest timestamp associated with the initial amessage (either on it or on an echo). Let u_1 be the earliest timestamp associated with the initial amessage. Let I_1 be the interval from u_1 to t_1. Let v_1 be the latest timestamp of a processor with a timestamp appearing in I_1. Let E_1 be the interval from u_1 to $v_1 + d$. Having defined E_i, let t_{i+1} be the latest timestamp associated with the deepest amessage with an associated timestamp at or earlier than $v_i + d$ and an associated timestamp after v_i. Let u_{i+1} be the earliest timestamp associated with this amessage. Let I_{i+1} be the interval from u_{i+1} to t_{i+1}. Let v_{i+1} be the latest timestamp of a processor with a timestamp appearing in I_{i+1}. Let E_{i+1} be the interval from u_{i+1} to $v_{i+1} + d$. Continue the process until all the timestamps are exhausted. For each i let R_i be the set of processors with timestamps appearing in the interval I_i. The cardinality of each R_i is at least c, since it contains the sender and all echoers of some amessage. By construction the sets R_i satisfy the hypotheses of Lemma 4.2. Thus the maximum index i is $k(n, c)$. The length of each interval E_i is $e + 2d$ plus an amount that corresponds to the difference between one timestamp and another by the same processor, where the timestamps are associated with different depth amessages. If the difference in depths is j then the difference in timestamps is less than $(j-1)d + 2d\gamma(j-1)$. Moreover, a later E_i involves deeper amessages. The latency is the maximum possible sum of the lengths of the E_i, which is bounded above by $k(n,c)e + (2k(n,c) + f - 1 + 2(\gamma(f)))d$. Thus we can take $\alpha = 2k(n,c) + f - 1 + 2\gamma(f)$.

The argument that the protocol achieves atomic broadcast is relatively simple. If a correct processor delivers a value, then either it received a corresponding amessage with depth $f+1$ or it sent a corresponding amessage to all other correct processors. In either case the other correct processors must also have delivered the corresponding value. □

Note that if there is no relative drift among correct clocks, then α is $2k(n,c) + 3f + 1$.

Theorem 4.7: There is an atomic broadcast protocol for n completely connected processors that tolerates arbitrary (Byzantine) failures in up to $f < \frac{n}{3}$ processors and has a latency of $2e + \beta$, where β is a term that depends only on n, f, and ρ.

Proof Sketch:

Since $f < \frac{n}{3}$, $k(n,c) = 2$. Following Srikanth and Toueg ([ST]) we convert the protocol of the proof of Theorem 4.5 to one that tolerates Byzantine failures by echoing the echoes with what we will call *aechoes* for authentication echoes. The echoes themselves serve to authenticate the amessages. The emessages are deemed authentic when $f + 1$ aechoes have been received directly; though, in order to send an amessage, there must be a total of c timestamps associated with each of the $c - 1$ echoes. This number guarantees that any processor will see the echo as authentic. It is straightforward to convert the protocol that uses signatures to one that does not. An easy overestimate gives $\beta = 3\alpha/2$. □

Theorem 4.8: There is an atomic broadcast protocol tolerant of timing faults at any f components for any $0 < f < n - 1$ that has a latency of $e + 3\gamma(f)d$.

Proof Sketch:

In our algorithm, each processor uses its clock to timestamp and send a message to each of its neighbors every d units of clock time. When one of these periodic messages is received, the receiver adds its own timestamp and the latest timestamp it has received from every other processor and returns the message to its sender. Finally, when a returned message is received it is again timestamped. Each processor maintains a FIFO protocol for its returned messages, never accepting receipt of a message out of order. It holds any out of order message until the missing message has arrived. Moreover, it refuses to receive any return that has taken longer than $2d$ units of time on its clock to make the round trip, permanently stopping the periodic messages to a neighbor that exhibits such a timing failure. The same treatment is given any processor that "returns" messages before they are sent.

In addition to the timestamps, these periodic messages carry no text on the first half of their cycle. But, when a processor originates an atomic broadcast, it timestamps the text with a time of origin (against which latency is measured) and then places the modified text on the next periodic message that it returns to each of its neighbors.

When a processor receives text with a time of origin only on a returned periodic message, it checks that the time of origin is between the time of receipt by the neighbor of the previous periodic message and that of the current periodic message. If not, then the message is discarded and the periodic messages to that neighbor are permanently halted; if so, then the time of return is added to the message text as a time of relay, the text is forwarded to the other neighbors on the next available periodic message, and the text of the atomic broadcast is scheduled for delivery at its time of origin plus $e + 3\gamma(f)d$.

When a processor receives a returned periodic message carrying a text with a time of origin and some times of relay, it checks that the last time of relay is between the time of receipt by the neighbor of the previous periodic message and that of the current periodic message. If not, then the message is discarded and the neighbor treated as above; if so, then the text of the atomic broadcast is scheduled for delivery at its time of origin plus $e + 3\gamma(f)d$.

Note that the elapsed time for any hop as measured on the clock of the receiver must be at most $3d$ because this is the maximum time between sending one periodic message and receiving the return of the next periodic message. If an atomic broadcast originates at a correct processor at local clock time t, then a correct neighbor receives the text by clock time $t + e + 3d$. Since processors may only suffer timing faults, a relaying processor can assume that each hop that the message took required no more than $3d$ on the local clock of the receiver. After f hops some correct processor must be encountered. Thus the latency is $e + 3\gamma(f)d$. □

Note that if there is no relative drift then the coefficient of d becomes $3(f + 1)$.

Corollary 4.9: For any $0 < f < n - 1$, the precision sensitivity of the class of timing failures is exactly 1. Thus the class of timing failures does not dominate the class of clock and generalized omission failures.

5 Summary

Coinciding upper and lower bounds for the time required to reach agreement are available only for the case of omission failures. In the past a large gap has separated upper and lower bounds for any class of failures containing timing failures. For any class of failures

containing both clock and generalized omission failures, we provide both improved upper and lower bounds.

We have proved tight bounds on precision sensitivity as a function of failure class; but we have still left wide gaps between upper and lower bounds for latency of atomic broadcast. We leave the closing of these gaps as open problems.

References

[CASD] F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," *proceedings, the 15th Int. Conf. on Fault Tolerant Computing,* (1985) 1-7. See also IBM Research Report RJ5244 (Revised 4/11/89).

[DHSS] D. Dolev, J. Halpern, B. Simons, and R. Strong, "Dynamic Fault-Tolerant Clock Synchronization," IBM Research Report RJ6722, March 3, 1989. See also "Fault-Tolerant Clock Synchronization," PODC-3, pp. 89-102, Vancouver, 1984.

[DM] C. Dwork and Y. Moses, "Knowledge and Common Knowledge in a Byzantine Environment I: Crash Failures," *Theoretical Aspects of Reasoning About Knowledge,* ed. J. Halpern, pp. 149-170, Monterey, 1986.

[DS] D. Dolev and R. Strong, "Authenticated Algorithms for Byzantine Agreement," *SIAM Journal of Computing,* vol. 12, no. 4, pp. 656-666, 1983.

[DRS] D. Dolev, R. Reischuk, and R. Strong, "Early Stopping in Byzantine Agreement," to appear in *JACM.* See also IBM Research Report RJ5406, December 2, 1986.

[FL] M. Fischer and N. Lynch, "A Lower Bound on the Time to Assure Interactive Consistency," *Information Processing Letters* 14:4, pp. 183-186, 1982.

[H] V. Hadzilacos, "Issues of Fault Tolerance in Concurrent Computations," *Ph.D. dissertation,* Harvard University, June 1984, Harvard University Department of Computer Science Technical Report 11-84.

[LSP] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems,* vol. 4, no. 3, pp. 382-401, July 1982.

[LT] N. A. Lynch, and M. R. Tuttle, *Hierarchical Correctness Proofs for Distributed Algorithms,* Technical Report MIT/LCS/TR-387, MIT, Lab. for Computer Science, April 1987.

[PSL] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *JACM* 27, pp. 228-234, 1980.

[NT] G. Neiger and S. Toueg, "Automatically Increasing the Fault-tolerance of Distributed Systems," *Proc. 7th ACM Symp. on PODC,* (1988) 248-262.

[ST] T. K. Srikanth, and S. Toueg, "Byzantine Agreement Made Simple: Simulating Authentication Without Signatures," *Distributed Computing* 2, pp. 80-94, 1987.