Inexact Agreement:
Accuracy, Precision, and Graceful Degradation

Stephen R. Mahaney*
AT&T Bell Laboratories

Fred B. Schneider+
Cornell University

ABSTRACT

An Inexact Agreement protocol allows processors that each have a value approximating $\hat{v}$ to compute new values that are closer to each other and close to $\hat{v}$. Two fault-tolerant protocols for Inexact Agreement are described. As long as fewer than 1⁄3 of the processors are faulty, the protocols give the required convergence; they also permit iteration and thus convergence to any desired precision. When between 1⁄3 and 2⁄3 of the processors are faulty, the protocols may not converge. However, then processors either detect that too many faults have occurred or the new values computed by processors remain close to each other and to $\hat{v}$. In this case, the divergence is bounded. Use of the protocols for clock synchronization in a distributed system is explained.

1. Introduction

It is frequently necessary for processors to agree on an approximation to some value. In certain applications, processors can use different values provided these values are close to each other and to the value being approximated. Clock synchronization [HSSD84] [LL84] [LM84] [MO83] is one application where such *Inexact Agreement* is acceptable. Other applications arise in industrial control where sensors measure some physical quantity and, under program control, actuators affect the process being monitored. Provided minor differences in the input used by processors running the industrial-control program do not result in radically different responses, Inexact Agreement can be used.

*Room 2C-323, AT&T Bell Laboratories, Murray Hill, N.J. 07974

+Department of Computer Science, Cornell University, Ithaca, N.Y. 14853.

Schneider is supported, in part, by NSF Grant DCR-8320274, by AT&T Bell Laboratories, and an IBM Faculty Development Award.

Inexact Agreement is weaker than Byzantine Agreement [LSP82] and, for a given degree of fault tolerance, is cheaper to achieve. This paper describes two new fault-tolerant protocols for achieving Inexact Agreement and explains how they can be used to implement fault-tolerant clock synchronization in a distributed system. The protocols improve on earlier work ([LM84] and [DLPSW83]) and illustrate some surprising performance trade-offs. In addition, the protocols exhibit graceful degradation—reasonable and predictable behavior—when as many as 2⁄3 of the processors are faulty. As long as fewer than 1⁄3 of the processors are faulty, the protocols permit iteration and convergence to a value. When between 1⁄3 and 2⁄3 of the processors are faulty, iteration may not lead to convergence. However, then processors will either detect that too many faults have occurred or the new values computed by processors will remain close to each other and to the value being approximated.

2. Problem Formulation

Consider a collection of N processors p, q, *etc.*, where each processor can reliably communicate over point-to-point links with all others and be assured of timely delivery of messages.[1] *Correct* processors execute their programs in a timely manner; *faulty* processors can exhibit arbitrary behavior. Let G be the set of correct processors and F be the set of faulty processors. Each processor p is assumed to have a value v_p, a real number approximating some quantity of interest $\hat{v}$. A value v_p is *correct* if $|v_p - \hat{v}| \le \kappa$ and considered *faulty* otherwise. Define G_{val} to be the set of correct processors with correct values and F_{val} to be the set of correct processors with faulty values. Thus,

[1]The reliable communications assumption is unnecessary, but simplifies the protocols and analysis.

$G = G_{val} \cup F_{val}$.

In an Inexact Agreement, each processor p computes a new value v_p'. An *Inexact Agreement* protocol is characterized in terms of the two quantities

$$\textit{Precision}: \max_{p,q \in G} | v_p' - v_q' | \text{ and}$$

$$\textit{Accuracy}: \max_{p \in G} | v_p' - \hat{v} | .$$

Precision measures how close new values are to each other; accuracy measures how close new values are to $\hat{v}$, the quantity being approximated.

The values in G_{val} are assumed to initially satisfy

$$\textit{Initial-Precision}: \max_{p,q \in G_{val}} | v_p - v_q | \le \delta, \text{ and}$$

$$\textit{Initial-Accuracy}: \max_{p \in G_{val}} | v_p - \hat{v} | \le \kappa.$$

The initial precision bound $\delta \le 2\kappa$ follows from the initial accuracy. Since accuracy and precision may not change in the same proportion, we describe them separately with two parameters, κ and δ. This permits, for example, describing the results of iterating our protocols, which enhance precision at the expense of accuracy.

An Inexact Agreement protocol with precision less than δ can be iterated to achieve even greater precision. Thus, Inexact Agreement protocols can be used to converge on a value.

Also, Inexact Agreement permits any correct processor p to compute a new value satisfying the precision and accuracy conditions, even if p starts with a faulty value. One consequence of this is that no restart protocol is required to integrate a new processor or reintegrate a repaired processor into a running ensemble. This is particularly useful in environments where a processor can erroneously detect that another processor is faulty. For example, if p uses a timeout to decide whether q is faulty, network congestion might cause p to decide that q is faulty when, in fact, q is not faulty and not even aware that p has decided it is. Because q is a correct processor, even though a fault has been attributed to it, it can continue running the protocol (perhaps with an incorrect value) without first executing a restart protocol.

A problem closely related to establishing Inexact Agreement is that of achieving *Approximate Agreement* [DLPSW83]. In Approximate Agreement, our accuracy requirement is replaced by *validity*, the stronger requirement that v_p' is in the range of the initial correct values at correct processors. We discuss the implications of this difference in §5.

3. Achieving Inexact Agreement

Our Inexact Agreement protocols are based on averaging over all values that are possibly correct according to a density requirement derived from the initial precision. The protocols are described in terms of multisets. Throughout, $V \cup W$ denotes the union of two multisets V and W. We use $\mathit{intvl}(V)$ for $[\min(V), \max(V)]$ an ordinary interval of the real line, $\mathit{width}(V)$ for $\max(V) - \min(V)$, and $\#(V, I)$ for the number of elements of V that have values in the interval I.

In a system with N processors that must tolerate $|F| + |F_{val}| \le m$ faults, a value v in a multiset V is *acceptable* if

$$(\mathit{oppE}\ s,\ f : s \le v \le f \quad f - s \le \delta : \#(V, [s, f]) \ge N - m).$$

An acceptable value might be correct since it is within δ of $N-m$ other values. However, two acceptable values need not be within δ of each other. A value is suspect if it is more than δ from other acceptable values. A value that is not acceptable is clearly faulty. The *acceptable set* A of a multiset V is the multiset containing all acceptable values in V; the *acceptable interval* of V is $\mathit{intvl}(A)$.

3.1. Fast Convergence Algorithm

Our first Inexact Agreement protocol, called the *Fast Convergence Algorithm* (FCA), uses a single round of message exchange. It uses an estimator function $e()$ (defined below) in place of faulty values. The protocol executed at processor p is:

Fast Convergence Algorithm:

1. Request and receive a value v_{rp} from every other processor r. Let V_p be the multiset of these values.
2. Construct the acceptable set A_p from V_p.
3. Compute $e(A_p)$.
4. Replace any v_{rp} in V_p that is not in A_p by $e(A_p)$.
5. Set v_p' to the average of the values in V_p.

A function $e(A)$ for a multiset A is an *estimator* provided it returns a value in $\mathit{intvl}(A)$. Thus, the average $e_{avg}(A)$, the median $e_{med}(A)$, and the midpoint $e_{mid}(A)$ are all estimators.[2] The choice of estimator affects precision and accuracy of FCA only when faulty values are detected and therefore does not affect the worst-case behavior of the algorithm, which is governed by undetectable errors. We discuss various estimators and analyze their performance in the Appendix.

Accuracy and Precision of FCA

To analyze the behavior of FCA, suppose processors p and q are correct.

Lemma 1: If $N \ge 3m+1$ and $|F| + |F_{val}| \le m$ then $\mathit{width}(\mathit{intvl}(A_p \cup A_q)) \le 2\delta$.

Proof: The proof is by contradiction. Since at most m values are faulty, correct values from correct processors must

[2] Using e_{avg} is equivalent to a simpler algorithm where steps (3), (4), and (5) are replaced by taking the average of the values in A_p.

be acceptable. Let $intvl(A_p \cup A_q)$ be $[v_s, v_f]$ and assume $v_f - v_s > 2\delta$. We show that $N \le 3m$, contradicting the hypothesis that $N \ge 3m+1$.

If $v_f - v_s > 2\delta$, then there are two disjoint subintervals, each of length δ, in $intvl(A_p \cup A_q)$: $L = [v_s, v_s+\delta]$ and $R = [v_f - \delta, v_f]$. Since v_s is acceptable for A_p (say), at least $N-m$ of the values in A_p are in L. Suppose $t_L \le m$ values from $A_p \cap L$ are faulty or are from faulty processors. Then at least $N-m-t_L$ correct values from correct processors are in A_p and also in A_q. Thus $A_p \cup A_q$ contains at least $2(N-m-t_L)+t_L = 2(N-m)-t_L$ values from L. Similarly, $A_p \cup A_q$ contains at least $2(N-m-t_R)+t_R = 2(N-m)-t_R$ values from R, where $t_R \le m$ values from $A_p \cap R$ are faulty or are from faulty processors.

Since L and R are disjoint, $A_p \cup A_q$ contains at least $4(N-m)-t_L-t_R$ values. Moreover, because A_p and A_q each contains at most N values, $4(N-m)-t_L-t_R \le 2N$, equivalently $2N \le 4m+t_L+t_R$. We have $t_L \le m$ and $t_R \le m$, so we can conclude $2N \le 4m+t_L+t_R \le 6m$. However, this implies $N \le 3m$, a contradiction. □

We can now prove

Theorem 2: If $N \ge 3m+1$, during execution of FCA there are a total of $t = |F|$ faulty processors and $|F| + |F_{val}| \le m$, then precision is bounded by $\frac{2t}{N}\delta$.

Proof: Suppose processor p is correct. Let v_{rp} be the value p obtains from processor r and let $\overline{v_{rp}}$ be defined by

$$\overline{v_{rp}} = \begin{cases} e(A_p) \text{ if } v_{rp} \text{ is not in } A_p \\ v_{rp} \text{ otherwise.} \end{cases}$$

Define $\overline{v_{rq}}$ for some other correct processor q similarly.

First, we prove that $|\overline{v_{rp}} - \overline{v_{rq}}| \le 2\delta$. By the definition of an estimator, if $\overline{v_{rp}} = e(A_p)$ then $\overline{v_{rp}}$ must lie in $intvl(A_p)$; otherwise, $\overline{v_{rp}} = v_{rp}$, which lies within $intvl(A_p)$ by definition. Similar reasoning allows us to conclude that $\overline{v_{rq}}$ lies within $intvl(A_q)$. Application of Lemma 1 yields $|\overline{v_{rp}} - \overline{v_{rq}}| \le 2\delta$.

Next, let v_p' be the average value computed by p in step (5) of the FCA protocol.

$$|v_p' - v_q'| = \left| \frac{1}{N}\sum_{r \in F \cup G} \overline{v_{rp}} - \frac{1}{N}\sum_{r \in F \cup G} \overline{v_{rq}} \right|,$$

$$= \frac{1}{N} \left| \sum_{r \in F \cup G} (\overline{v_{rp}} - \overline{v_{rq}}) \right|,$$

$$= \frac{1}{N} \left| \sum_{r \in F} (\overline{v_{rp}} - \overline{v_{rq}}) \right|$$

since for $r \in G$, $|\overline{v_{rp}} - \overline{v_{rq}}| = 0$,

$$\le \frac{1}{N} \sum_{r \in F} |\overline{v_{rp}} - \overline{v_{rq}}|,$$

$$\le \frac{1}{N} 2t\delta$$

since $|F| = t$ and, by the argument above, $|\overline{v_{rp}} - \overline{v_{rq}}| \le 2\delta$. □

Theorem 3: If $N \ge 3m+1$ and during execution of FCA, $t = |F| + |F_{val}| \le m$, then accuracy is bounded by $\kappa + \frac{t}{N}\delta$.

Proof: Every correct value v_{rp} received by p from a correct processor r, by definition, satisfies $|\hat{v} - v_{rp}| \le \kappa$. Other values v_{rp}, that are acceptable must be within δ of a correct value from a correct processor. Thus, such a value must satisfy $|\hat{v} - v_{rp}| \le \kappa+\delta$. This also means that the acceptable interval $intvl(A_p)$ is contained in $[\hat{v}-(\kappa+\delta), \hat{v}+(\kappa+\delta)]$ and because an estimator $e(A_p)$ must lie within $intvl(A_p)$, $|\hat{v} - e(A_p)| \le \kappa+\delta$.

The difference between $\hat{v}$ and v_p' the new value computed by p is given by

$$|\hat{v} - v_p'| = \left| \hat{v} - \frac{1}{N} \sum_{r \in G_{val} \cup F_{val} \cup F} v_{rp} \right|,$$

$$\le \frac{1}{N} \sum_{r \in G_{val} \cup F_{val} \cup F} |\hat{v} - v_{rp}|,$$

$$= \frac{1}{N}\Big(\sum_{r \in G_{val}} |\hat{v} - v_{rp}|$$

$$+ \sum_{r \in F_{val} \cup F} |\hat{v} - v_{rp}| \Big),$$

$$= \frac{1}{N}\Big(\sum_{r \in G_{val}} \kappa + \sum_{r \in F_{val} \cup F} \kappa+\delta \Big)$$

substituting based on the arguments given above,

$$\le \frac{1}{N}((N-t)\kappa + t(\kappa+\delta)),$$

$$= \kappa + \frac{t}{N}\delta$$

which was to be proved. □

The bounds of Theorems 2 and 3 are worst when $t = m$. Then precision is bounded by $\frac{2m}{3m+1}\delta < \frac{2}{3}\delta$ and accuracy is bounded by $\kappa + \frac{m}{3m+1}\delta < \kappa + \frac{\delta}{3}$. Moreover, these bounds are tight, as shown by

Theorem 4: There exist scenarios in which the worst-case precision and accuracy are achieved by FCA.

Proof: A scenario that achieves worst-case precision is given in the following table. Each entry in the table gives the number of instances of each value received by processor p or q. Thus, the table states that m processors give the value $-\delta$ to p, but the value $+\delta$ to q. All the values are acceptable.

value:	$-\delta$	$-\delta/2$	$0=\hat{v}$	$\delta/2$	δ
V_p:	m	0	$N-m$	0	0
V_q:	0	0	$N-m$	0	m

A scenario that achieves worst-case accuracy is:

value:	$0=\hat{v}$	κ	$\kappa+\delta$
V_p:	0	$N-m$	m

The calculations are straightforward. □

Although the loss of accuracy in FCA (and CCA of §3.2) is not desirable, it is manageable. Devices like clocks and sensors typically exhibit inaccuracy when operating. Usually, when such a device proves insufficiently accurate, we simply replace it with a more accurate device. If the loss in accuracy (at worst from κ to $\kappa+\delta/3$) that arises when FCA must cope with m faulty values is not acceptable, we can handle the problem in the same way as with a single device: obtain devices that are more accurate. The result is a system that achieves the original desired accuracy, is fault-tolerant, and has very high precision.

Graceful Degradation of FCA

The performance of FCA degrades gracefully when the total number of faulty processors and correct processors with faulty values exceeds m. For $|F| + |F_{val}| = t$, where $m < t < N-m$, a correct processor executing FCA will either

- detect and report that more than m faults have occurred, or
- compute a new value that satisfies reasonable accuracy and precision constraints (given below).

When $t \ge N-m$, no assertion can be made about the values computed by correct processors, because then there are enough faulty processors for their values, no matter how inaccurate, to form an acceptable set.

A correct processor can construct a non-empty acceptable set if $t \le m$. Thus, whenever a processor cannot construct such an acceptable set, it can report that more than m faults must have occurred. If at least $m+1$ processors report such an occurrence, we know there must have been more than m faults.

We now derive bounds on precision and accuracy for processors that construct acceptable intervals even though $m < t < N-m$. The analysis is different from that in Theorems 2 and 3 because it is now possible that some of the correct values from correct processors are not acceptable. The following scenario illustrates this possibility for $N=5$ processors, $m=1$, and $t=2$ faults. Correct values are marked with an $*$.

value:	0	$\delta/2$	$\delta=\hat{v}$	$3\delta/2$	2δ
V_p:	2	1*	1*	1*	0
V_q:	0	1*	1*	1*	2

Notice that p considers only the values at 0, $\delta/2$, and δ acceptable and therefore replaces the correct value at $3\delta/2$ with the estimator; q has symmetric behavior. Thus, in bounding the precision of FCA when $m < t < N-m$, we must account for differences between correct values acceptable at one processor and replaced by the estimator at another. This means that the choice of estimator can affect the worst-case behavior of the protocol.

In the above scenario the value at δ is acceptable at both p and q. It is possible that *no* correct value will be acceptable to both p and q. The following scenario with $N=7$, $m=2$, and $t=4$ illustrates this possibility.

value:	0	δ	$3\delta/2=\hat{v}$	2δ	3δ
V_p:	4	1*	1*	1*	0
V_q:	0	1*	1*	1*	4

Here, the values at 0 and δ only are acceptable to p; q accepts values at 2δ and 3δ. There is no correct value acceptable at both p and q and there is a correct value at $3\delta/2$ that both replace by the estimator. Thus, in bounding precision of FCA, we must consider differences between the values of estimators at two correct processors.

The first scenario above illustrates sharing of correct values. A value v_r is *shared* by p and q if r is a correct processor, v_r is a correct value, and every acceptable value in $A_p \cup A_q$ is within δ of v_r. Sharing need not occur, as illustrated in the second scenario above; in this case, $width(intvl(A_p \cup A_q)) \le 3\delta$. When sharing does occur, $width(intvl(A_p \cup A_q)) \le 2\delta$.

We show below that precision is worse when sharing does not occur, but for certain values of N, m, and t. sharing must occur. Our analysis is based on the worst possible choice of estimators. The results of a more detailed analysis when e_{mid} is used as the estimator are also mentioned below.

In the following, we assume

There are t faulty processors or correct processors with faulty values (i.e., $|F \cup F_{val}| = t$);

All processors in $F \cup F_{val}$ and their corresponding values are termed *bad*;

$N = 3m+k$ where $k = 1, 2,$ or 3, so that m is chosen as large as possible;

And, t satisfies $m < t < N-m$.

Lemma 5: Suppose N, m, and t are as in the assumptions above and that processors p and q construct acceptable intervals with $s \ge 1$ shared values. Then, precision is bounded by $\frac{N-s}{N} 2\delta$.

Proof: Define v_{rp}^{-} and v_{rq}^{-} as in Theorem 2. By assumption, every value in $A_p \cup A_q$ is within δ of some shared value. Thus, $|v_{rp}^{-} - v_{rq}^{-}| \leq 2\delta$ whether r is bad and gives different values to p and q, or whether r is correct and v_{rp} is accepted and v_{rq} is rejected and replaced by an estimator (or *vice versa*).

Let S denote the set of processors whose values are shared, and R denote all other processors. Similar to the inequalities of Theorem 2, we have

$$|v_p' - v_q'| = \left| \frac{1}{N} \sum_{r \in S \cup R} v_{rp}^{-} - \frac{1}{N} \sum_{r \in S \cup R} v_{rq}^{-} \right|,$$

$$\leq \frac{1}{N} \sum_{r \in R} |v_{rp}^{-} - v_{rq}^{-}|$$

since shared values add 0 to the differences,

$$\leq \frac{1}{N}(N-s)2\delta$$

since $|S| = s$ □

Lemma 6: Suppose N, m, and t are as in the assumptions and that processors p and q construct acceptable intervals with no shared values. Then, precision is bounded by $\frac{N+2t+2m}{N}\delta$.

Proof: First, we establish bounds on the differences $|v_{rp}^{-} - v_{rq}^{-}|$. The cases to consider are: (case i) when r is bad; (case ii) when r is correct and at least one of the values v_{rp}, v_{rq} is acceptable; and (case iii) when r is correct and both values are rejected. Let F, RA and RR denote the sets of processors satisfying these three cases, respectively.

For a value to be acceptable it must be within δ of a correct value from a correct processor. This is because there are fewer than $N-m$ values from $F \cup F_{val}$, so the acceptable interval cannot consist solely of bad values and therefore every acceptable value must lie in a δ-width interval containing some correct values.

It follows that for any r, $|v_{rp}^{-} - v_{rq}^{-}| \leq 3\delta$. When r is bad (case i) or r is correct but both values are rejected (case iii), we may use this bound. If r is a correct processor and v_{rp} is accepted at p but v_{rq} is rejected at q (or vice versa), then $|v_{rp}^{-} - v_{rq}^{-}| \leq 2\delta$ (case ii).

Recall $t = |F \cup F_{val}|$. Since p constructs an acceptable interval, there must be at least $N-m-t$ correct processors in RA satisfying (ii). Since q forms an acceptable interval, RA contains at least an additional $N-m-t$ more values, by the same argument. So $|RA| \geq 2(N-m-t)$. Then we have $|RR| \leq N-t-2(N-m-t)$.

We now have

$$|v_p' - v_q'| = \left| \frac{1}{N} \sum_{r \in F \cup RA \cup RR} v_{rp}^{-} - \frac{1}{N} \sum_{r \in F \cup RA \cup RR} v_{rq}^{-} \right|,$$

$$\leq \frac{1}{N} \Big(\sum_{r \in F \cup RR} |v_{rp}^{-} - v_{rq}^{-}|$$

$$+ \sum_{r \in RA} |v_{rp}^{-} - v_{rq}^{-}| \Big),$$

$$\leq \frac{1}{N} \big([(N-t-2(N-m-t))+t]\, 3\delta$$

$$+ [2(N-m-t)]\, 2\delta \big)$$

choosing as many values as possible in $F \cup RR$,

$$= \frac{(N+2t+2m)}{N} \delta.$$

□

We now have bounds on precision depending on whether sharing occurs. The following gives conditions on N, m, and t that characterize when sharing must occur.

Lemma 7: Suppose N, m, and t are as in the assumptions and that processors p and q construct acceptable intervals. Define $f = t - m$. Then there exist scenarios without sharing if and only if $N \leq 3m+f$. If sharing must occur, then at most s values must be shared where s is given by:

If $N = 3m+2$ and $t = m+1$, then $s = 1$;
If $N = 3m+3$ and $t = m+1$, then $s = 2$;
If $N = 3m+3$ and $t = m+2$, then $s = 1$.

Proof: In a scenario where p and q construct acceptable intervals without sharing, the t bad values accepted by p are within δ of at least $l = N-m-t$ correct values from correct processors. The t bad values accepted by q must be near at least l correct values from correct processors that are distinct from those accepted at p since there is no sharing. Thus, there must be at least $t+2l$ processors. So, if no sharing is possible, then

$$\begin{array}{ll} & t+2l \leq N \\ \text{iff} & t+2N-2m-2t \leq N \\ \text{iff} & N \leq 2m+t \\ \text{iff} & N \leq 3m+f \end{array}$$

For the other direction, if $N \leq 3m+f$, then we have $t+2l \leq N$. Let $u = N-(t+2l)$. A scenario in which there is no sharing can now easily be constructed.

value:	0	δ	$3\delta/2 = \hat{v}$	2δ	3δ
V_p:	t	$N-m-t^*$	u^*	$N-m-t^*$	0
V_q:	0	$N-m-t^*$	u^*	$N-m-t^*$	t

This proves the first assertion about no sharing.

If sharing must occur, in the first case above, we show at most 1 value must be shared. Acceptable intervals can be formed with disjoint sets of m correct values from correct processors as in the following scenario where p accepts values in $[-\delta,\ 0]$ and q accepts values in $[0,\ +\delta]$:

value:	$-\delta$	$-\delta/2$	$0 = \hat{v}$	$\delta/2$	δ
V_p:	t	m	1	m	0
V_q:	0	m	1	m	t

The number of processors is sufficient, since

$$\begin{aligned} t+m+1+m &= t+2m+1 \\ &= (m+1)+2m+1 \\ &= 3m+2 \\ &= N. \end{aligned}$$

The other cases are proved similarly. □

The above lemmas are summarized in the following.

Theorem 8: Let N, m, and t be as in the assumptions and suppose processors p and q construct acceptable intervals. Precision of FCA is bounded by

$$\frac{N-s}{N} 2\delta < 2\delta$$

if sharing at s values occurs, and by

$$\frac{(N+2t+2m)}{N}\delta < 3\delta$$

if sharing does not occur. Lemma 7 gives the conditions on when sharing must occur. When sharing is not guaranteed, worse precision is attained without sharing.

Proof: Lemmas 5, 6 and 7 establish all but the last claim. To prove that, it suffices to show

$$\frac{N-s}{N} 2\delta \le \frac{(N+2t+2m)}{N}\delta.$$

Since $s \ge 1$ and $t \ge m+1$, this follows from algebra. □

We now address accuracy and graceful degradation for FCA.

Theorem 9: Let N, m, and t be as in the assumptions and suppose processor p constructs an acceptable interval. Accuracy of FCA is bounded by $\kappa + \frac{m+t}{N}\delta < \kappa+\delta$

Proof: As in Theorem 3, first we bound $|\hat{v} - \overline{v_{rp}}|$. For correct values v_{rp}, $|\hat{v} - v_{rp}| \le \kappa$ due to initial accuracy. Since bad values that are accepted must be within δ of a correct value, for r bad, we have $|\hat{v} - \overline{v_{rp}}| \le \kappa+\delta$. Finally, if v_{rp} is correct but not acceptable, then since the estimator must lie in the range of accepted values, $|\hat{v} - \overline{v_{rp}}| \le \kappa+\delta$. Let CA, F, and CR denote the sets of processors that furnished correct, acceptable values; bad values; and correct, but not acceptable values, respectively. There are at most t processors in F and at least $N-m-t$ in CA. Thus $|CA| \ge N-m-t$, so $|F \cup CR| \le m+t$. We can now bound accuracy.

$$\begin{aligned} |\hat{v} - v_p'| &= \Big|\hat{v} - \frac{1}{N}\sum_{r \in CA \cup F \cup CR} \overline{v_{rp}}\Big|, \\ &\le \frac{1}{N}\sum_{r \in CA \cup F \cup CR} |\hat{v} - \overline{v_{rp}}|, \\ &= \frac{1}{N}\Big(\sum_{r \in CA} |\hat{v} - v_{rp}| + \sum_{r \in F \cup CR} |\hat{v} - v_{rp}|\Big), \\ &\le \frac{1}{N}([N-m-t]\kappa + [m+t](\kappa+\delta)) \end{aligned}$$

by the bounds above and choosing CA as small as possible,

$$= \kappa + \frac{m+t}{N}\delta < \kappa+\delta$$

since $m+t < N$. □

For the case where e_{mid} is used as the estimator, we can prove [MS85]:

Theorem 10: Let N, m, and t be as in the assumptions and suppose processors p and q construct acceptable intervals. Suppose that e_{mid} (midpoint) is used as the estimator. Then precision $|v_p' - v_q'|$ is bounded by:

If sharing not necessary, $\frac{\delta}{N}(N+2t+m)$.
If $N = 3m+2$ and $t = m+1$, $\frac{\delta}{N}(N+t-1)$.
If $N = 3m+3$ and $t = m+1$, $\frac{\delta}{N}(N+t-2)$.
If $N = 3m+3$ and $t = m+2$, $\frac{\delta}{N}(N+t-1)$.

and accuracy is bounded by:

$$\kappa + \frac{2t+m}{2N}\delta < \kappa + \frac{5}{6}\delta.$$

Iteration for Convergence

When FCA is iterated to achieve higher precision, there are even stronger constraints on when a value is acceptable assuming only $t < N-m$. If we expect $t \le m$ faults, then on the i^{th} iteration we can expect to achieve precision $(2/3)^i\delta$. If the loss of precision characterized by Theorem 8 (or 10) occurs at the i^{th} iteration then it will not be possible to construct an acceptable interval using $(2/3)^i\delta$ (in place of δ). In addition, from the initial set of values a processor p receives it can construct an interval VI_p in which $\hat{v}$ must lie. When FCA is iterated, at least $N-m$ values must lie within $\kappa+\delta/3$ of VI_p after each iteration. Therefore, if the loss of accuracy characterized by Theorem 9 (or 10) occurs, this too can be detected by p. In short, if FCA is iterated and $m < t < N-m$ then either this situation will be detected or (only) the last iteration of the protocol can introduce the error characterized by Theorems 8 and 9 (or 10).

3.2. Crusaders Convergence Algorithm

The precision of FCA is limited by faulty processors that can be "two-faced". A faulty value appearing at the low end of the acceptable interval used by one processor and at the high end of the acceptable interval used by another will cause the averages for the acceptable sets at the processors to differ, thereby reducing precision.[3] We can reduce this problem by having processors agree on some of the values included in the acceptable set. We avoid using Byzantine

[3]Faulty values that fall outside the acceptable interval don't reduce precision because the estimator maps these to a potentially correct value.

Agreements to accomplish this because of the number of rounds required. Instead, every processor uses a cheaper Crusaders Agreement to disseminate its value to other processors.

A *Crusaders Agreement* [D82] allows a designated processor, called the *transmitter*, to disseminate a value in such a way that:

CRU1: All correct processors that do not "know" that the transmitter is faulty agree on the same value.

CRU2: If the transmitter is correct then all correct processors agree on its value.

Thus, Crusaders Agreement potentially partitions processors into three classes: those that are faulty, those that are correct and "know" that the transmitter is faulty, and those that are correct and have agreed among themselves on a value from among those the transmitter sent. Crusaders Agreement is simple and inexpensive to implement in a network where $N \geq 3m+1$ and our reliable communications assumption holds. A 2-round protocol is given in [D82]:

(1) The transmitter sends its value to all other processors.

(2) Each processor sends the value it has received from the transmitter to all processors (including itself).

(3) Each processor sifts through the values it received in step (2) to identify a set of at most m suspicious processors that, if faulty, could account for inequalities among the values. If after ignoring values received from suspicious processors a single value remains, then agree on it; otherwise, decide that the transmitter is faulty.

The *Crusaders Convergence Algorithm* (CCA) below is based on FCA, but exploits a Crusaders Agreement to prevent faulty processors from adversely affecting precision. The protocol is described for processor p.

Crusaders Convergence Algorithm:

1. Broadcast value v_p.
   Broadcast all values received.
   Using Crusaders Agreement, decide on the value v_{rp} of every other processor r.
   Let V_p be the multiset of these values.
2. Construct the acceptable set A_p from V_p.
3. Compute $e(A_p)$.
4. Replace every value in V_p that is not also in A_p by $e(A_p)$.
   (If the Crusaders Agreement decided that transmitter r was faulty then v_{rp} is replaced by $e(A_p)$.)
5. Set v_p' to the average of the values in V_p.

Accuracy and Precision of CCA

The analysis of CCA is similar to that for FCA. The primary difference is the structure of the worst-case scenario. Since a Crusaders Agreement is used, if a faulty processor r causes a correct processor p to decide on a value v_{rp} then any other correct processor q will either decide on the same value or will decide r is faulty and use the estimator for r's value. Thus, the worst-case analysis of CCA must consider $|v_{rp} - e(A_q)|$, rather than differences due to different values from the same faulty processor.

In the analysis below, we use the median e_{med} as the estimator; other choices we have investigated do not give as good precision. First, we bound differences between acceptable values and e_{med}.

Lemma 11: If $N \geq 3m+1$, then for correct processors p and q with acceptable sets A_p and A_q, every value v_{rp} in A_p satisfies $|v_{rp} - e_{med}(A_q)| \leq \delta$.

Proof: Since $\max(A_p) - \min(A_p) \leq 2\delta$ by Lemma 1, any $v_{rp} \in A_p$ is either in $[\min(A_p), \min(A_p)+\delta]$ or in $[\max(A_p)-\delta, \max(A_p)]$. If we can show that

$$e_{med}(A_q) \in [\min(A_p), \min(A_p)+\delta] \text{ and} \tag{11.1}$$

$$e_{med}(A_q) \in [\max(A_p)-\delta, \max(A_p)], \tag{11.2}$$

then we can establish by (11.1) or (11.2) respectively that v_{rp} differs from $e_{med}(A_q)$ by at most δ, and the Lemma will be proved.

We prove (11.1); the proof of (11.2) is similar. Consider the following multisets, enumerated in increasing sorted order:

$$A_p \cap A_q = \{g_1, g_2, \cdots\}$$

$$A_p = \{x_1, x_2, \cdots\}$$

$$A_q = \{y_1, y_2, \cdots\}$$

Let m_p values be acceptable at p but not at q; i.e. $m_p = |A_p - A_q|$. Observe that for any k, $1 \leq k \leq |A_p \cap A_q|$:

$$g_k \leq x_{k+m_p}, \tag{11.3}$$

$$x_k \leq g_k, \tag{11.4}$$

since A_p contains $A_p \cap A_q$ and has at most m_p more elements than $A_p \cap A_q$. Similar properties hold for the enumeration of A_q.

To establish (11.1), we must show $\min(A_p) \leq e_{med}(A_q)$ and that $e_{med}(A_q) \leq \min(A_p)+\delta$. To see that $\min(A_p) \leq e_{med}(A_q)$, first note that $N-m+m_q \leq |A_q|$. Then

$$\begin{aligned} & m_q+1 \leq \lfloor (N-m+m_q)/2 \rfloor \\ \text{iff} \quad & 2(m_q+1) \leq N-m+m_q \\ \text{iff} \quad & m_q+2 \leq N-m \end{aligned}$$

which is always true. Thus, we have

$$\begin{aligned} \min(A_p) &= x_1 \text{ by the definition of } x_1, \\ &\leq g_1 \text{ by (11.4)}, \\ &\leq y_{m_q+1} \text{ by (11.3)}, \end{aligned}$$

$$\leq y_{\lfloor |A_q|/2 \rfloor}$$

by the above inequality that $m_q \leq \lfloor |A_q|/2 \rfloor$ and the fact that the y_i are sorted,

$$\leq e_{med}(A_q)$$

by the definition of median.

We now establish that $e_{med}(A_q) \leq \min(A_p)+\delta$. First, note that $|A_q| \leq N-m_p$, because for each of the m_p values v_{rp} in A_p-A_q, CRU1 of the Crusader Agreement guarantees that r cannot contribute a different value to A_q. Now we have

$$\lceil |A_q|/2 \rceil \leq \lceil (N-m_p)/2 \rceil \qquad (11.5)$$

Next, note that since $3m+1 \leq N$ and $m_p \leq m$ we can conclude $2m+m_p < N$. Rearranging terms, we get $N-m_p < 2N-2m-2m_p$, and by algebra we obtain

$$\lceil (N-m_p)/2 \rceil \leq N-m-m_p \qquad (11.6)$$

Finally, we have

$e_{med}(A_q) \leq x_{\lceil |A_q|/2 \rceil}$ by the definition of median,

$\leq x_{\lceil (N-m_p)/2 \rceil}$ by (11.5),

$\leq g_{\lceil (N-m_p)/2 \rceil}$ by (11.4),

$\leq g_{N-m-m_p}$

by (11.6) and the fact that the g_i are sorted,

$\leq x_{N-m}$ by (11.3),

$\leq x_1+\delta$

since x_1 being acceptable implies at least $N-m$ values within δ of x_1.

$$= \min(A_p)+\delta$$

as was to be proved. □

We can now prove

Theorem 12: If $N \geq 3m+1$ and there are $t \leq m$ faulty processors during execution of CCA, then precision is bounded by $\frac{t}{N}\delta$.

Proof: Suppose processor p is correct. Let v_{rp} be the value p obtains from processor r and let $\overline{v_{rp}}$ be defined by

$$\overline{v_{rp}} = \begin{cases} e(A_p) \text{ if } v_{rp} \text{ is not} \in A_p \\ v_{rp} \text{ otherwise.} \end{cases}$$

Define $\overline{v_{rq}}$ similarly.

Next, let v_p', v_q' be the average values computed by p and q in step (5) of the CCA protocol.

$$|v_p' - v_q'| = \left| \frac{1}{N}\sum_{r \in F \cup G} \overline{v_{rp}} - \frac{1}{N}\sum_{r \in F \cup G} \overline{v_{rq}} \right|,$$

$$\leq \frac{1}{N}\sum_{r \in F \cup G} |\overline{v_{rp}} - \overline{v_{rq}}|,$$

$$= \frac{1}{N}\sum_{r \in F} |\overline{v_{rp}} - \overline{v_{rq}}|$$

since for a correct processor r, $|\overline{v_{rp}} - \overline{v_{rq}}| = 0$,

$$= \frac{1}{N}\Bigl(\sum_{r \in A_p - A_q} |v_{rp} - e_{med}(A_q)|$$

$$+ \sum_{r \in A_q - A_p} |e_{med}(A_p) - v_{rq}| \Bigr),$$

$$\leq \frac{1}{N}(|A_p - A_q|\,\delta + |A_q - A_p|\,\delta)$$

by Lemma 11,

$$\leq \frac{1}{N} t\delta$$

since $|A_p - A_q| + |A_q - A_p| \leq t$ by CRU1. □

Theorem 13: If $N \geq 3m+1$ and there are $t \leq m$ faulty processors and correct processors with faulty values, then accuracy is bounded by $\kappa + \frac{t}{N}\delta$.

Proof: Same as Theorem 3. □

Notice that since $N \geq 3m+1$, the worst-case precision is $\delta/3$, half that of FCA, and the worst-case accuracy is the same as FCA. When FCA is iterated twice, the worst-case precision is $4\delta/9$, clearly inferior to the $\delta/3$ precision achieved by CCA. Thus, at least in this case, it is better to use a second round of message exchange for agreement on the input to a convergence algorithm rather than iterating the convergence itself.

The worst-case bounds for CCA are tight, as is shown by

Theorem 14: There exist scenarios in which the worst-case precision and accuracy bounds are achieved for CCA.

Proof:

A scenario that achieves worst-case precision is:

value:	$-\delta$	$-\delta/2$	$0=\hat{v}$	$\delta/2$	δ
V_p:	0	m	$N-2m$	m	0
V_q:	0	m	$N-3m$	m	m

The scenario that achieves worst-case accuracy is the same as given in Theorem 4. □

Graceful Degradation of CCA

CCA can be modified to degrade gracefully when $|F| + |F_{val}| < N-m$ because the above Crusaders Agreement protocol degrades gracefully. When $|F| < N-m$, instead of CRU1 and CRU2, the Crusaders Agreement protocol above establishes

CRU′: If the transmitter is correct, then each correct processor either decides on the transmitter's value or decides that the transmitter is faulty.

This is because if $|F \cup F_{val}| < N-m$, there are at least $m+1$ correct processors. Thus, from a correct transmitter q, a correct receiver p will receive at least $m+1$ copies of q's message. The processor p might receive sufficiently many copies of q's message to agree on it (if enough faulty processors broadcast it correctly). However, faulty processors can send at most $N-m-1$ copies of a different message. Thus, p will either agree on the message q sent or determine (erroneously) that q is faulty.

Therefore, when the Crusaders Agreement protocol is run with $t < N-m$ faults, if the transmitter is correct then other correct processors are potentially partitioned into two classes: those that have decided on the transmitter's value and those that have decided (erroneously) that the transmitter is faulty. And, if the transmitter is faulty then we can make no assertion about the values correct processors decide on.

In a system with $N-m-1$ faulty processors there are at most $N-m-1$ faulty transmitters, so each correct processor can agree on at most $N-m-1$ faulty values. As with FCA, if a processor cannot construct a non-empty acceptable set, it has detected that the number of faults exceeds m and can announce this fact and/or then use FCA on the values it received in the first round of the Crusaders Agreement. The graceful degradation of FCA ensures graceful degradation here. If a correct processor can construct an acceptable interval then this interval might contain some of the $N-m-1$ faulty values from faulty transmitters. However, for a faulty value to be part of the acceptable set at a processor, it must be within δ of $N-m-1$ other values, which means it must be within δ of a correct value. This permits the accuracy and precision of the new values computed by correct processors to be bounded in the same way as for FCA.

Theorem 15: Suppose that $N = 3m+k$ for $k=1$, 2, or 3, and that $t = |F \cup F_{val}|$ satisfies $m < t < N-m$. If p and q form acceptable intervals in the execution of CCA, then accuracy and precision have the same bounds as in Theorems 8, 9 and 10.

Proof: The arguments of Theorems 8, 9 and 10 apply without modification. □

Thus, precision is at worst 3δ and accuracy is bounded by $\kappa+\delta$. And, as with FCA, iterating the protocol to achieve higher precision only improves the chances that the excessive number of faults will be detected.

4. Application to Clock Synchronization

Our Inexact Agreement algorithms can be used to implement fault-tolerant protocols for synchronizing clocks. The approach we take here is based on the *interactive convergence* algorithm of [LM84].[4] There, processor clocks are periodically brought together using an Inexact Agreement protocol. FCA has better precision than the Inexact Agreement protocol used in [LM84][5], so simply replacing the Inexact Agreement part of that protocol yields an improved clock resynchronization protocol. CCA, which has an even higher precision than FCA, cannot be used as the Inexact Agreement part of the protocol without modifications (discussed below), however. This is because the Crusaders Agreement in CCA must be altered to allow processors to agree on clocks. Unlike simple values, different copies of a clock may not be equal due to uncertainty in network delivery delays and clock drift. The Crusaders Agreement Algorithm of §3.2 allows processors to agree on copies of values, only.

Below, we first sketch a clock synchronization protocol based on FCA. This provides the background necessary for understanding the modifications required to CCA so that it can be used for clock synchronization. We next sketch that. A detailed analysis of the performance of our clock synchronization protocols is not given here, since it differs from that in [LM85] only in the use of a different value for precision.

4.1. Using FCA to Synchronize Clocks

Each processor p is assumed to have a real-time clock which we model as a function c_p from real time to clock time. Correct real-time clocks satisfy

$$\textit{Rate Restriction}: \quad \left| \frac{dc_p(t)}{dt} - 1 \right| < \frac{\rho}{2}$$

for some given maximum drift rate ρ. Correct real-time clocks can drift apart; a clock synchronization protocol periodically resynchronizes them. Faulty clocks may not satisfy the Rate Restriction and a clock synchronization protocol must be able to tolerate this.

Given a resynchronization procedure that brings correct clocks to within some bound $CONV(\delta) < \delta$ of each other provided they initially read within δ of each other, the clock at each processor p must be resynchronized every R seconds (according to p's clock), where R is chosen to be small enough so that clocks on correct processors do not drift farther than δ seconds apart during this interval. Let p be the processor with the slowest correct clock. It will take longest for R seconds to elapse on p's clock, and therefore this interval defines the maximum that clocks on correct processors can drift apart during an R second interval on any of them. R seconds according to p's clock is $\frac{2R}{(2-\rho)}$ real seconds because p's clock runs at $1-\frac{\rho}{2}$ p-seconds per second. According to the Rate Restriction, the fastest and slowest clocks drift apart at ρ clock-seconds per real-second. Thus, the maximum that correct clocks drift apart over an R

[4] It is also possible to replace the Approximate Agreement protocol used in the clock resynchronization protocol of [LL84] with FCA or CCA. The protocols of [HSSD84] and [MO83], however, cannot benefit from using FCA or CCA because they are not based on convergence.

[5] The ratio of the precision of FCA to the Inexact Agreement protocol used in [LM84] is 2/3.

second interval on any of them is $\frac{2R\rho}{(2-\rho)}$. Since the previous execution of the clock synchronization protocol brings correct clocks within $CONV(\delta)$ of each other, we must chose R so that

$$CONV(\delta) + \frac{2R\rho}{2-\rho} \leq \delta, \text{ or}$$

$$R \leq \frac{(\delta - CONV(\delta))(2-\rho)}{2\rho}$$

If it were possible for one processor to read the clock of another, the desired resynchronization protocol would simply be for each processor p to execute an Inexact Agreement algorithm with precision $CONV(\delta)$ at R second intervals according to its clock. However, it is usually not possible for one processor to read the clock of another—at least not without introducing some error in the values read, due to uncertainty in network delays and different clock rates.

A technique originally proposed by [LM84] allows one processor to read the clock of another. Suppose the minimum delay (according to the clock at any correct processor) incurred in sending a message between any pair of processors is λ and that $\lambda+\varepsilon$ is the maximum delay incurred. Thus, ε is the uncertainty in delivery time for a message (according to the clock at any correct processor). A processor p can thus read the value of another processor q's clock by executing

> **send** "time?" **to** q;
> **receive** c_{qp} **from** q **timeout after** $2(\lambda+\varepsilon) + \gamma$;
> **if** timed-out **then** $c_{qp} := \infty$;
> $\Delta_{qp} := c_p(now) - \lambda - c_{qp}$

where γ is the maximum length of time (according to p's clock) it takes for q to process the "time?" request made by p. After executing this protocol, $c_p(now)-\Delta_{qp}$ is an approximation of $c_q(now)$, processor q's clock when p's clock reads *now*. Thus, p can read q's clock whenever necessary simply by reading its own clock and subtracting Δ_{qp}.

We are now almost ready to use FCA to resynchronize clocks. However, in the analysis of FCA we assumed in step (1) that all correct processors receive the same value from a given correct processor. Because of the way processors read clocks, this is no longer the case: $c_p(now)-\Delta_{rp}$, r's clock at real time *now* according to p, and $c_q(now)-\Delta_{rq}$, r's clock at real time *now* according to q, can differ due to errors introduced by the way clocks are read. This error can be bounded, though. The maximum by which any two correct clocks can drift apart in L seconds according to any correct clock was derived above to be $\frac{2L\rho}{2-\rho}$. An additional error of ε is incurred due to uncertainty in message delivery delays. Thus, after L seconds on the clock of any correct processor, the maximum error E in "reading" the clock on another correct processor is

$$E = \varepsilon + \frac{2L\rho}{2-\rho}.$$

Typically, L will be small, since it is the amount of time that elapses between steps (1) and (3) of the FCA protocol, so E will be small.

Fortunately, FCA achieves good precision even when correct processors receive slightly different values from a given correct processor. If a value read by two correct processors differs by at most D then we redefine an acceptable value to be one within $\delta+D$ of $N-m$ other values. Now, we can prove:

> **Lemma 16:** If $N \geq 3m+1$, there are $t \leq m$ faulty processors, and a value read by two correct processors differs by at most D, then $width(intvl(A_p \cup A_q) \leq 2(\delta+D)$.

Proof: Identical to Lemma 1, except $\delta+D$ is substituted for δ. □

Using this, we can prove:

> **Theorem 17:** If $N \geq 3m+1$, during execution of FCA there are $t \leq m$ faulty processors, and a value as read by two correct processors differs by at most D, then precision of FCA is bounded by $\frac{(N+t)D+2t(\delta+D)}{N}$.

Proof: Suppose processors p and q are correct and define v_{rp}, $\overline{v_{rp}}$, v_{rq}, and $\overline{v_{rq}}$ as in Theorem 2.

First, we prove that $|\overline{v_{rp}}-\overline{v_{rq}}| \leq 2(\delta+D)$. By the definition of an estimator, if $\overline{v_{rp}} = e(A_p)$ then v_{rp} must lie in $intvl(A_p)$; otherwise, $\overline{v_{rp}} = v_{rp}$, which lies within $intvl(A_p)$ by definition. Similar reasoning allows us to conclude that $\overline{v_{rq}}$ lies within $intvl(A_q)$. Application of Lemma 16 yields $|\overline{v_{rp}}-\overline{v_{rq}}| \leq 2(\delta+D)$.

Next, let v_p' be the average value computed by p in step (5) of the FCA protocol.

$$|v_p' - v_q'| = \left| \frac{1}{N}\sum_{r \in F \cup G} \overline{v_{rp}} - \frac{1}{N}\sum_{r \in F \cup G} \overline{v_{rq}} \right|,$$

$$= \frac{1}{N}\left(\left| \sum_{r \in F \cup G} \overline{v_{rp}} - \sum_{r \in F \cup G} \overline{v_{rq}} \right| \right),$$

$$\leq \frac{1}{N}\left(|G|\, D + \left| \sum_{r \in F} \overline{v_{rp}} - \sum_{r \in F} \overline{v_{rq}} \right| \right)$$

since for a correct processor r, $|\overline{v_{rp}} - \overline{v_{rq}}| \leq D$,

$$\leq \frac{1}{N}\left(|G|\, D + \left| \sum_{r \in F} \overline{v_{rp}} - \overline{v_{rq}} \right| \right),$$

$$\leq \frac{1}{N}\left(|G|\, D + \sum_{r \in F} |\overline{v_{rp}} - \overline{v_{rq}}| \right),$$

$$\leq \frac{1}{N}(N-t)D + 2t(\delta+D)$$

since by the argument above, $|\overline{v_{rp}} - \overline{v_{rq}}| \leq 2(\delta+D)$ and $|F| = t$ and $|G| = N-t$. □

Thus, by choosing $D = E$ we get

$$CONV(\delta) \leq \frac{(N-m)E+2m(\delta+E)}{N},$$

which is close to $2\delta/3$ since E is likely to be very small. It is now possible to compute R and periodically use FCA to resynchronize clocks.

One problem remains, however. We can think of each execution of the resynchronization protocol by a given processor as starting a new clock at that processor. The precision of FCA ensures that the value for the i^{th} clock at each correct processor will be within $CONV(\delta)$ provided the values of the $i-1^{st}$ clocks are used in the computation, since the $i-1^{st}$ clocks remain within δ of each other for their first R seconds. To ensure that a processor only uses values from the $i-1^{st}$ clocks when setting its i^{th} clock, we must further restrict the resynchronization period R. In particular, we must make R sufficiently large so that a processor with a fast clock does not attempt to start its i^{th} clock until all other processors have started their $i-1^{st}$ clocks. Let q be the processor with the fastest correct clock and p be the processor with the slowest correct clock. If p and q start out perfectly synchronized, then there is a difference of

$$\frac{2R}{2-\rho} - \frac{2R}{2+\rho}$$

$$= \frac{4R\rho}{4-\rho^2}$$

real seconds between R clock seconds on p's clock and on q's clock. However, at the start of a resynchronization period, the clocks at p and q can differ by as much as $CONV(\delta)$ seconds. The worst case is when p is behind by $CONV(\delta)$ seconds, because then the difference (in real seconds) between when p and q reach the next resynchronization point will be the largest. By choosing R so that it is greater than this interval we ensure that the processor with the slowest clock has started its $i-1^{st}$ clock by the time the processor with the fastest clock needs that value in order to start its i^{th} clock:

$$R \geq \frac{2(R+CONV(\delta))}{2-\rho} - \frac{2R}{2+\rho}.$$

4.2. Using CCA to Synchronize Clocks

A processor p can send a copy of q's clock to processor r by computing Δ_{qp} as above and then sending Δ_{qp} to r. Upon receipt of this value, r can compute $c_r(now)-\Delta_{qp}-\Delta_{pr}$ to approximate the time according to q's clock. However, if r has read q's clock directly it could also use $c_r(now)-\Delta_{qr}$ to approximate the time according to q's clock. These values might not be equal, since they have associated error.

The unfortunate consequence of this inequality is that it is no longer possible in step (3) of the Crusaders Agreement protocol to use equality of (approximate) clock values as a way to identify suspicious processors. For a transmitter q, step (1) of the agreement protocol permits r to compute Δ_{qr} based on c_{qr}, received directly from q. Step (2) of the protocol permits processor r to compute Δ_{qr}^p, an approximation of the clock at q, based on receiving Δ_{pr} from r: $\Delta_{qr}^p = \Delta_{pr}-\Delta_{qp}$. These copies of r's clocks at q can differ by at most a small error E, as given above. Thus, we modify step (3) of the Crusaders Agreement protocol for a transmitter q and use this error bound when trying to identify suspicious processors.

(3′) A processor r sifts through all the values it has received to identify a set of suspicious processors containing every processor s for which Δ_{qr}^s differs from Δ_{qr} by more than E. If it is possible to find a set of suspicious processors of size m or smaller, use Δ_{qr} as the value; otherwise decide that the transmitter is faulty.

This allows CCA to replace FCA in the clock resynchronization procedure described in §4.1, and, due to the higher precision of CCA, closer clock synchronization is achieved each time the protocol is run.

5. Comparison with Other Work

FCA was inspired by and is based upon the interactive convergence algorithm developed by Lamport and Melliar-Smith [LM84] for clock synchronization. There, each processor p first acquires a value v_q from every other processor q, computes the multiset of values $\{v_q : |v_q - v_p| \leq \delta\}$, and then sets v_p' to the average of this multiset. When there are $t \leq \lfloor (N-1)/3 \rfloor$ faults, the precision of this algorithm is $\frac{3t}{N}\delta \leq \frac{N-1}{N}\delta$ and there are scenarios in which this is actually achieved. Thus, FCA provides 2/3 better precision and the same accuracy. In addition, correct processors exhibit graceful degradation regardless of the number of faults.

Two algorithms for achieving Approximate Agreement are described in [DLPSW83] and applied to clock synchronization in [LL84]. Neither algorithm exhibits graceful degradation should the actual number of faults experienced exceed $m = \lfloor (N-1)/3 \rfloor$. In the first algorithm, which we call DLPSW-mid, the highest and lowest m of the values received are discarded and then the midpoint of the remaining values is computed. By discarding these $2m$ values, the remaining $N-2m$ and the midpoint thereof are guaranteed to be bounded by correct values. DLPSW-mid has precision of $\delta/2$ and accuracy of κ. Consequently, when the actual number of faults is greater than $N/4$ DLPSW-mid gives greater accuracy and precision than FCA; when there are fewer than $N/4$ faults, FCA gives better precision, but not better accuracy. And, as the number of faults approaches 0, the accuracy of FCA approaches that of this algorithm.

The second algorithm proposed in [DLPSW83], which we call DLPSW-avg, computes the average after discarding the highest and lowest $m = \lfloor (N-1)/3 \rfloor$ values. For $t \leq m$ faults, the accuracy of this algorithm is κ and the precision is $\frac{t}{N-2m}\delta$. FCA gives better precision but worse accuracy for all $0 < t \leq m$. In fact, FCA provides 2/3 better precision than DLPSW-avg in the limit as m goes to infinity.

For a full and fair comparison of our methods with those in [DLPSW83], the differences between Inexact and Approximate Agreement must be examined. Our basic view is that computer and control systems are built using devices (clocks, sensors, *etc.*) that, even when operating correctly, exhibit errors that we have modeled as accuracy and precision. We seek algorithms that can tolerate some maximum number of faults, and we are satisfied with accuracy comparable to that of a single correct device. Precision is the measure of how good such an algorithm is. Graceful degradation is a measure of robustness.

The algorithms in [DLPSW83] are based on the assumption that at most m of N processors (or sensors) are faulty and achieve validity—that new values at correct processors are in the range of the initial correct values. Here, too, precision is a measure of how good such an algorithm is. However, validity is a stronger requirement than accuracy; it seems to mimic Byzantine Agreement, rather than model error in systems. Indeed, [DLPSW83] contains lower bound proofs for Approximate Agreement that resemble the corresponding proofs for Byzantine Agreement.

It has been noted that to a person with a hammer, every problem looks like a nail. With our view in terms of bounded error, the hammer is knowledge that $N-m$ of the values are correct and lie in some δ interval; we screen out provably incorrect values. And, our basic result is that such screening gives convergence—and quite quickly—when $N \geq 3m+1$. For [LM84] the hammer is knowing that correct values are separated by at most δ; a correct processor p will average over values within δ of its own. This gives convergence for up to m faults, $N \geq 3m+1$ and guarantees graceful degradation for up to $N-1$ faults. The hammer for [DLPSW83] is the maximum number of faults, m; before taking a midpoint or average, they screen out m values from each extreme leaving values that are either correct or bounded by correct values. Their basic result is that convergence is achieved—although not quite as quickly—when $N \geq 3m+1$.

Perhaps the greatest surprise in all this is that such different approaches give such similar precision. In addition, the comparison of FCA with DLPSW-mid and DLPSW-avg offers two data points to suggest that there is a tradeoff between accuracy and precision. FCA is more precise than DLPSW-mid when the actual number of faults is bounded by $N/4$ and always more precise than DLPSW-avg, but the precision is achieved at a loss of accuracy.

6. Summary

We have presented two protocols to achieve Inexact Agreement. They give better precision, but possibly worse accuracy, than previous work [DLPSW83] [LM84]. They also exhibit graceful degradation—reasonable and predictable behavior—when as many as $2/3$ of the processors are faulty. No previous has addressed this issue. We have shown how these protocols can be used to synchronize clocks in a distributed system.

Acknowledgments

We wish to thank A. Aho, O. Babaoglu, and M. D. McIlroy for comments on an earlier version of this paper and L. Lamport and N. Lynch for helpful discussions concerning convergence and clock synchronization.

References

[D82] Dolev, D. The byzantine generals strike again. *J. of Algorithms 3*, 14-30 (1982).

[DLPSW83] Dolev, D., N.A. Lynch, S.S Pinter, E.W. Stark, and W.E. Weihl. Reaching approximate agreement in the presence of faults. *Proc. Third Symposium on Reliability in Distributed Software and Database Systems*, Oct. 1983, IEEE Computer Society, 145-154.

[HSSD84] Halpern, J., B. Simons, R. Strong, and D. Dolev. Fault-tolerant clock synchronization. *Proc. of the Third ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1984, ACM, 89-102.

[LL84] Lundelius, J. and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Proc. of the Third ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1984, ACM, 75-88.

[LM84] Lamport, L. and P.M. Melliar-Smith. Byzantine clock synchronization. *Proc. of the Third ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1984, ACM, 68-74.

[LM85] Lamport, L. and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *JACM 32*, 1 (Jan. 1985), 52-78.

[LSP82] Lamport, L., R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. on Prog. Lang. and Sys. 4*, 3 (July 1982), 382-401.

[MO83] Marzullo, K., and S.S. Owicki. Maintaining the time in a distributed system. *Proc. of the Second ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1983, ACM, 295-305.

[MS85] Mahaney, S.R. and F.B. Schneider. Failing with Grace. In preparation.

[S84] Schneider, F.B. Byzantine Generals in action: Implementing fail-stop processors. *ACM Trans. on Computer Systems 2*, 2 (May 1984) 145-154.

Appendix: Choosing an Estimator

When faulty values can be detected, estimators are largely responsible for imprecision in FCA and CCA. Estimators cause imprecision in two ways. The first occurs when two processors p and q use an estimator in place of some third processor r's value. Here,

Estimator-estimator Disagreement: $| e(A_p) - e(A_q) |$

causes imprecision, since p uses a different value than q does for r's value. The second source of imprecision occurs when p uses an estimator in place of r's value, but q finds r's value to be acceptable. Here,

Estimator-value Disagreement: $| v_{rp} - e(A_q) |$

is a source of imprecision.

An estimator that minimizes estimator-estimator disagreement may not minimize estimator-value disagreement and *vice versa*, as is shown in the table below.

In order to select an estimator intelligently, information is needed about whether correct processors will agree that a given faulty value is faulty. Since detected faulty values are replaced by the estimator, if processors are likely to agree when a value is faulty, then an estimator that minimizes estimator-estimator disagreements should be chosen. For example, if, with high probability, processors halt in response to a failure [S84] then use of timeouts allows the failure of a processor to be detected by all others; *average*, which minimizes estimator-estimator disagreement, is therefore a good choice.

If, on the other hand, processors are not likely to agree on whether a given value is faulty, then an estimator that minimizes estimator-value disagreements should be selected. For example, noise in a communications line usually results in detectably faulty values being sent to a few processors, while the remaining processors receive correct (and identical) values. We should therefore attempt to minimize estimator-value disagreement here, since there will be more of them than estimator-estimator disagreements. Thus, *midpoint* would be a good choice of estimator because it has low estimator-value disagreement.

Bounds on Maximum Disagreement		
	$\| e(A_p) - e(A_q) \|$	$\| v_{rp} - e(A_q) \|$
e_{avg}	$\frac{2m}{N}\delta$	$\frac{N+2m}{N}\delta$
e_{med}	δ	2δ
e_{mid}	δ	$\frac{3}{2}\delta$