Elections in the Presence of Faults

Michael Merritt †
AT&T Bell Laboratories

1. Introduction

The news media often bombards the public with forecasts of election results. Polls predict, sometimes years in advance; exit polls are more accurate, and unofficial tallies tend to be closer to the final results. If close elections are disputed, it may take the courts weeks to determine the actual outcome of an election. If the election is nearly unanimous, however, a few disputed votes can have no outcome on the final results. The time at which the final results may be known with certainty thus depends upon the accuracy of the forecast (the number of disputed votes), and the closeness of the election.

In a network of processors, determining the outcome of an election quickly could have many advantages. For example, early motivation for agreement problems was that of several processors reading a single sensor, and voting on the value read [PSL80]. Under normal conditions, each processor would read the same value, and the election would be unanimous. Accurate forecasts would free the processors to act on the election results, even before the final returns were obtained. Only under increasingly unlikely conditions of multiple failure would there be sustained disagreement about the election results.

† Address: AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974.

This paper explores the problem of forecasting election results quickly and accurately in a network of unreliable processes. It presents synchronous protocols which forecast results with known accuracy, under a variety of failure assumptions. It is really a close examination of the Interactive Consistency problem [PSL80], which is essentially an election, with the new feature of exploring the amount of partial information available during the course of the election.

The protocols for restrictive failure models are similar to known solutions to the Interactive Consistency problem--for less restrictive (byzantine) failures, the protocols are original, and introduce the notion of *witnesses*. This technique may be of use in the design of other protocols. All the protocols are simple to present and understand, and straightforward proofs of correctness are presented.

The results presented here raise more questions than they answer, however. For the less restricted failure model, lower bounds are conjectured which match the behavior of the protocols presented, but known techniques for obtaining lower bounds do not extend directly. 'Finer' techniques for examining the executions of protocols are needed. Even extending the protocols presented is a hard problem--showing that optimal use has been made of the witnessing technique would solve a fifteen year-old graph conjecture, if only in a restricted case!

The remainder of this paper is organized as follows. The rest of this section introduces the Election problem more carefully, and discusses its relation to Interactive Consistency in more detail. In the second section, protocols for restrictive Stopping-Fault models are discussed. Authenticated protocols for byzantine failures are addressed in the third section, and the paper closes with a brief discussion.

1.1. Network Assumptions

The scenario consists of n processors, a subset of whom (the *voters*) each have a local value they wish to broadcast. Each processor is trying to collect the election results--a vote for each voter.

We make several assumptions about the communications network. It is fully connected, so that any processor may send messages to any other. The network never loses or alters messages, and reliably identifies the sender of any message to the receiver. We bound the time it takes the net to deliver a message, and assume that the processors are themselves synchronized, so that protocols can be executed in synchronous *rounds*. During each round every processor p will:

a) Send some messages (any number, to any processor),
b) Receive all the messages sent to p this round, and
c) Evaluate a *decision function*, which returns p's current best guess at the final election results.

A synchronous election protocol tells each processor what messages to send each round (on the basis of messages received previously), and what decision function to use. Individual processors may fail (in a variety of ways) during a protocol execution, but in all cases the following assumption is made.

k-Fault Assumption

The number of processors which may suffer faults during a protocol's execution is bounded by a constant k. We consider protocols which are resistant to three different types of failures: two types of *Stopping−Faults*, in which failed processors behave correctly until failure and then halt permanently, and *Byzantine Failures*, in which failed processors may behave arbitrarily, though we assume a secure authentication (signature) scheme.

We are interested in protocols which take few rounds in the worst case; we would also like certain 'well-behaved' properties to hold. Typically, this will mean that wrong forecasts can occur only for a failed processor's vote, and that the number of disputed votes is small.

1.2. Elections and the Interactive Consistency Problem

If we were not concerned about forecasting results, the Election problem we have described reduces to the Interactive Consistency problem [PSL80]. By allowing processors to commit outcomes early, accurate forecasting of election results may be important to specific applications, especially those in which close elections are rare and time is important.

Another reason for examining the Election problem is the light it sheds (or shade it produces) on the question of lower bounds on the time needed to reach agreement. A series of results have established ever stronger lower bounds for Byzantine Agreement and for the Interactive Consistency problem ([DLM82], [DS82], [FL82], [LF84]). These results depend upon a particular technique which does not extend to the Election problem. Whether or not the lower bounds conjectured here are correct, new techniques will be required to obtain lower bounds for the Election problems considered here.

2. Stopping-Fault Elections

Stopping−Fault describes processors with well-behaved failure characteristics. Stopping-Fault processors behave correctly until failure, after which they do nothing. The system failure characteristics depend upon the atomic actions of the processors--the indivisible steps during which no failure can occur. For example, if broadcasting is an atomic action, it is trivial to design an election protocol producing exact returns after only a single round. We consider the sending of a single message to a single recipient an atomic act. Depending upon a processor's control over the message-sending mechanism, there may still be different types of system behavior. In particular, we consider two cases, one in which each processor can determine the order in which messages are to be sent each round, and another in which this order is not determined. These alternatives are referred to as *Sequenced* or *Unsequenced* Stopping-Fault processors, respectively. The difference affects the inferences about failure that processors can make--if Alice fails to receive a scheduled message from Bob, she knows he has failed. In the sequenced case, she knows that anyone scheduled to receive a message from Bob after her will not receive it, but in the unsequenced case everyone else may receive a message from Bob that round.

The next section presents a simple election protocol, and shows that it permits accurate forecasting as the election progresses.

2.1. The Stopping-Fault Election

Messages sent in this protocol have the form (p,v), where p is a processor name and v is a vote. The rounds of execution are numbered from 0 on, so that round j is actually the $j+1$'st round of execution. The protocol follows.

Stopping-Fault Election

```
Round 0
        Every voter $p$ broadcasts $(p,v)$,
        where $v$ is $p$'s vote.

Round $j$, $1 \le j \le k$

        Every processor broadcasts
        any new messages received
        during the previous round.

Decision Procedure

        Each processor $p$ does the following,
        for every processor $q$;

        If a message of the form $(q,v)$
        has been received, choose $v$ as $q$'s vote.
        Choose *error* otherwise.
```

Because processors are Stopping-Fault, the decision procedure will always be uniquely determined (there cannot be two messages of the form (p,v) and (p,u) sent during the same protocol execution).

Theorem 1

For Stopping-Fault (Sequenced or Unsequenced) processors, executions of the Notarized protocol satisfy the following properties:

i) If p does not fail, every correct processor will choose p's value every round.

ii) After round k, all correct processors have chosen the same values.

iii) After round j, $0 \le j < k$, values chosen for at most $k-j$ different processors are different from those eventually chosen.

Proof

Property (i) is clearly true. To see that (ii) and (iii) are true, note that there can only be disagreement on a value for a processor that fails during round 0. Note also that every processor receives the same messages during any round in which noone fails (all transmissions are broadcasts). More importantly, everyone has seen the same set of messages after such a round, and noone will later change any value. Then (ii) follows immediately. For any value to change after round j, there must therefore have been at least one failure each round through round j. This means at most $k-j$ processors could have failed during round 0, since there are at most k failures in all. □

For Unsequenced Stopping-Faults, Theorem 1 is as strong as possible; there are executions in which there is disagreement on as many as $k-j$ values after round j. It was conjectured for a time that this was a general lower bound for Unsequenced Stopping-Faults. As with all the failure models considered here, showing that there may be k disagreements after the first round is trivial, and $k+1$ rounds are necessary for total agreement, by the known bound on Interactive Consistency [FL84]. Cynthia Dwork has shown, however, that agreement on at least $k-2$ values is possible after round 1, when $k>2$ [D84]. Protocols achieving this small additional agreement are fairly complex, and it is not known if the techniques used can be extended to provide further early agreement.

As we are about to see, the simple Stopping-Fault protocol allows earlier agreement in the Sequenced Stopping-Fault model.

2.2. Sequenced Stopping-Fault Processors

If processors can schedule the order in which messages are sent during a given round, the Stopping-Fault Election can actually guarantee earlier agreement on more votes, although $k+1$ rounds are still required in the worst case (by a simple extension of the lower bound for Interactive Consistency in the unsequenced case). Exactly how much better is not known--deciding which schedules allow the strongest inferences is a difficult problem.

Suppose every round each processor p sends first to $p+k+1$ (*mod n*), then to $p+k+2$ (*mod n*) and so on. Call this scheduling a *$k-$sequencing*. We can show the following.

Theorem 2

Let $k=(r^2+3r)/2$, and $n>(r+1)k$. Then after two rounds of the Stopping-Fault protocol, Sequenced Stopping-Fault processors using k-sequencing will disagree on no more than $k-r$ votes.

The proof of this theorem shows that if more than $k-r$ processors fail in the first round, there will be agreement on more than $k-r$ votes after the second round. The k-sequencing is crucial to the argument.

3. Authenticated Protocols for Byzantine Elections

In this section we present protocols for systems in which processors may suffer *byzantine* failures--failures which can lead to arbitrary, even malicious behavior. The only limitation we make is that each processor can authenticate messages (e.g., using a digital signature scheme) in an unforgeable way. An authenticated protocol is a *Byzantine Election* protocol if it satisfies conditions i-iii of Theorem 1, in the face of byzantine failures. When there is only one voter, these requirements are those for Byzantine Agreement on the voter's value, with the restriction that agreement be reached in at most $k+1$ rounds, and with the addition of 'guesses' at every round. It is known that no such protocol takes less than $k+1$ rounds to reach agreement, in the worst case ([DS82],[DLM82]).

3.1. Witnesses

The idea behind both protocols presented below is that each processor p has an associated set of *witnesses*: these witnesses sign and forward messages containing values signed by p. More importantly, only forwarded information from appropriate witnesses is used to decide on a processor's vote. Witnesses are assigned *a priori*, in such a way that k liars (faulty processors) cannot simultaneously lie about their own values and about the values of other liars. In the first protocol these witnesses do not themselves participate in the election; like notary publics, they participate only as witnesses. In the second protocol, processors act as both voters and witnesses.

3.2. Notarized Election

As indicated above, the processors in this protocol are divided into two sets: the *voters* and the *witnesses*. Any processor may suffer from a byzantine failure, though no processor may forge another's signature. We require there to be at least $2k$ witnesses--thus, at least k witnesses are correct.

Definitions. A value signed by voter i is an *$i-$vote*. A witness's signature of an i-vote is an *affadavit* for that i-vote.

If there are $2k$ witnesses, the following protocol satisfies the Byzantine Election requirements for any number of voters.

Notarized Election

```
Round 0:
    Each voter signs and broadcasts his vote.
    Witnesses send no messages.

Round $j$, $1 \le j \le k$:

    Voters send no messages.

    Every witness $w$ does the following:

    For every voter $i$

        Any $i$-vote with $j-1$ or more
        different affadavits is valid.

        Sign any new valid $i$-votes,
        producing new affadavits.

        Broadcast every valid $i$-vote or affadavit
        for a valid $i$-vote that was not
        broadcast by $w$ in earlier rounds.


Decision Procedure for Round $j, 0 \le j \le k$

    If processor $p$ has received exactly one $i$-vote
    with at least $j$ different affadavits
    (counting his own, if $p$ is a witness)
    then $p$ chooses the value signed as $i$'s vote.
    Otherwise $p$ chooses error as $i$'s vote.
```

Lemma 1

In any execution of the Notarized Election, if any processor q receives an affadavit with p's signature for some i-vote v by the end of round j, $1 \le j < k$, then one of the following is true:

i) p is faulty, or

ii) every processor receives k different affadavits for v by the end of round $j+1$.

Proof

If p is not faulty, then the affadavit received by q was originally broadcast by p, along with the i-vote and at least $i-1$ other affadavits, during some round i, $i \le j$. There must be at least k correct witnesses, each of whom receive this broadcast, and for whom v is thus valid during round $i+1$. Thus (ii) is true. □

Lemma 2

In any execution of the Notarized Election, if any correct processor p changes its guess for some processor i after round j, $0 \le j < k$, then at least j witnesses are faulty.

Proof

The lemma is trivial for $j=0$. Suppose $j>0$, and some correct processor p chooses a value for i after round j that is different from some value chosen at a later round. Then p has chosen a value v or *error*, and later changes the value to another. There are three cases;

Case 1: Processor p chooses v after round j, and later agrees on a different value u. Since (ii) of Lemma 1 is false, the j signers of the affadavits for v are faulty.

Case 2: Processor p chooses v after round j, and during r agrees on *error*. Possibly p never receives more than j affadavits for v or any other i-vote, and chooses *error* for this reason. Then the j signers of the affadavits he now holds are faulty, by Lemma 1. The other possibility is that p later receives more than j affadavits for at least two i-votes, one of which contains a value u different from v. Either all of the witnesses signing these affadavits are faulty or choose the first correct processor q to sign u. This must happen after round j, so q has at least j affadavits from faulty processors.

Case 3: Processor p chooses *error* after round j, and later chooses a value v. Suppose p chooses *error* because he has no adequately witnessed i-votes. Later p has more than j affadavits for v. The witnesses for these affadavits are either all faulty, or some correct witness first signs v after round j, so that at least j of the affadavits seen by this witness are signed by faulty processors. The other possibility is that p chooses *error* after round j because of having at least two adequately witnessed i-votes. Since p later has only one adequately witnessed i-vote, by Lemma 1 the witnesses for at least one of the two are faulty. □

Theorem 3

If there are at least $2k$ witnesses, the Notarized Election is a Byzantine Election protocol.

Proof

First we must show that there is never any disagreement on correct processor's votes. Each correct processor i broadcasts an i-vote containing his value during round 0, and signs no other values at all during the protocol. Thus there can be at most one valid i-vote received by any processor in any round. Since at least k of the witnesses are not faulty, they will each receive this i-vote during round 0, and rebroadcast it with their affadavit during round 1. Thereafter, every processor will have received at least k different affadavits for that i-vote, so the appropriate value will be chosen as i's vote each round through round k.

Next, we show that all correct processors have reached byzantine agreement on every vote after round k. Suppose two processors r and s disagree on i's vote. Then one, say r, must have chosen a value v, while s has chosen *error* or a different value, u. In any case, one of the two has received some i-vote with k affadavits and the other has not. Then all k of the witnesses signing the affadavits are faulty--otherwise, they would have forwarded the i-vote and affadavits to both r and s, not just one of the two. But i must be faulty, too, since there is never disagreement on a correct processor's value. This is a total of $k+1$ faults, contradicting the k-fault hypothesis.

Finally, we show that after round j, for $0 \leq j < k$, final agreement is reached on $max(0, |voters| + j - k)$ values. If any value at all is changed after round j, there are j faulty witnesses, by Lemma 2. Thus there are at most $k-j$ faulty voters. Since values for correct processors are never changed, the result follows. □

3.3. Mutually Verified Election

The Notarized protocol required the presence of $2k$ processors who act only as witnesses. This may be inconvenient or expensive for many applications. What if every processor wishes to participate in the election? The following presents a Byzantine Election protocol for $n > k(k+1)$ processors, in which all processors vote.

Definitions. Numbering the processors from 1 to n, for $n > k(k+1)$, we designate the processors $i+1, \ldots, i+k+j (mod\ n)$ as $j-witnesses$ for processor i. As before, an i-vote is a value signed by i. A $j-$*affadavit* for an i-vote is the signature by a j-witness of that i-vote. For each j from 1 to k, construct G_j, the digraph with n nodes and $k+j$ arcs from each node to it's j-witnesses.

Mutually Verified Election

```
Round 0:
    Each processor signs and broadcasts his vote.

Round $j, 1 \leq j \leq k$:

    Every processor $w$ does the following.

    For every processor $i$:

        Any $i$-vote or more different
        $j$-affadavits is *valid*.

        Sign any valid $i$-vote that was not valid
        in previous rounds.

        If there are no valid $i$-votes, or $w$ is not
        a $j$-witness for $i$, send nothing.

        Else broadcast all of the $i$-votes
        and their $j$-affadavits that have
        not been broadcast by $w$ in earlier rounds.
```

Decision Procedure for Round $j, 0 \leq j \leq k$

If processor p has received exactly one i-vote with j or more different j-affadavits (counting his own, if p is a j-witness for i) then p chooses the value signed as i's vote. Otherwise, p chooses *error* as i's vote.

Lemma 3

The shortest cycle in G_j has length greater than $k-j+1$.

Proof

Picture the nodes of G_j on a circle--each node connected to the next $k+j$ in the clockwise direction. Starting at any node, the farthest around the circle one can get following a single edge is $k+j$ nodes. Any cycle must go completely around the circle, so the number of nodes in the cycle, times this maximum 'distance' covered by a single edge, must be at least n. Since $n > k(k+1)$, the result follows.

Lemma 4

Among any subset S of k processors, at most $k-j$ have j or more j-witnesses in S, for $1 \leq j \leq k$.

Proof

The lemma follows if any k-node subdigraph of G_j has at most $k-j$ nodes with outdegree j or more. Suppose that $k-j+1$ nodes in some k-node subdigraph G' of G_j have outdegree at least j. Then there is a cycle of length at most $k-j+1$ in G', and so in G_j, contradicting Lemma 3.

Lemma 5

In any execution of the Mutually Verified Election, if any correct processor p changes its guess for some processor i after round j, $0 \leq j < k$, then at least j of the j-witnesses for i are faulty.

Proof

By a case analysis similar to that in Lemma 2.

Theorem 4

If $n > k(k+1)$, the Mutually Verified Election is a Byzantine Election protocol.

Proof

Every correct processor signs and broadcasts a single value during round 0; furthermore, each correct processor has at least j correct j-witnesses. Thus, there will never be any disagreement on correct processors' values.

Now we argue that Byzantine Agreement is reached on every vote at the end of round k. Two processors r and s can disagree on some value for i only if one, say r, has received an i-vote with k k-affadavits that s has not received. Then each of these k k-witnesses must be faulty, along with i itself, contradicting the k-fault hypothesis.

Finally, we must show that after round j, for $0 \leq j \leq k$, final agreement is reached on at least $n+j-k$ values. Suppose this is not the case; that after round j of some execution, for each processor i among some set S of $k-j+1$ processors there is a correct processor who will later choose a different value for i than the one currently chosen. Since the values chosen for correct processors never change, each processor in S is faulty. By Lemma 3, the subdigraph of G_j generated by S has no cycles. Thus there must be a processor i in S such that every j-witness for i is not in S. By Lemma 4, i has at least j faulty j-witnesses; the j faulty witnesses for i, together with the $k-j+1$ members of S, contradict the k-fault assumption. $\square$

3.4. Other Byzantine Election Protocols

Neither the Notarized Election nor the Mutually Verified Election will work for general networks of processors--the one requires $2k$ witnesses, the other requires that there be more than $k(k+1)$ voters. While one can trade-off witnesses against the total number of voters by combining tricks from both protocols, this will not produce a completely general algorithm. It is possible that an execution-dependent choice of witnesses (replacing the fixed k-witnesses of the Mutually Verified protocol) might produce a completely general protocol--this possibility is under investigation.

Another possiblility is that a better fixed assignment of witnesses might be possible in the Mutually Verified protocol. Translated to a graph problem, and restricted to just the second round of the protocol, we need to know the following; is there a digraph on $n \leq k(k+1)$ vertices, such that each node has outdegree $k+1$ and there is no cycle of size less than or equal to k? A negative answer is a restricted case of a fifteen year-old conjecture ([BT81])!

Conjecture

If D is a digraph of degree at most dr, such that every vertex has outdegree r or more, then D contains a cycle of length at most d.

The best known result is that there is a cycle of length at most $d+2500$ [CS83]. This is an indication of the difficulties encountered in designing good election protocols, and in attempting to discover strong lower bounds to match.

4. Lower Bounds

This paper has conjectured the optimality of the two Byzantine Election protocols--that no protocols can forecast more accurate returns more quickly, in the byzantine failure model. For the first round, it is obvious there may be disagreement on as many as k votes, and after k rounds the bounds for Interactive Consistency show there is at least one vote of disagreement, in the worst case. It is interesting to note that the processors cannot all know on which votes they disagree--for then they could all pick a default value, and achieve Interactive Consistency too early.

These bounds for the first and kth round also hold for the Stopping-Fault failure models, but in these cases more votes can be accurately forecast early in the election. How many more votes can be forecast, and how early? Theorem 2 suggests about $\sqrt{k}$ more votes can be forecast after two rounds, but nothing more is known.

This is as far as known results and techniques go. Lower bounds for Interactive Consistency depend crucially on the requirement that *total* agreement be reached among the participating, failure-free processors. Crudely, this means that one can reason: "If Alice saw X and Bob saw Y, then if Bob saw Y and Carol saw Z, Carol would have to choose what Alice chose when she saw X, since Bob will choose the same values and they must both agree with him." This reasoning is not valid in our setting, as both Alice and Carol are free to disagree a little with Bob, and thence perhaps completely with each other. Thus a relation which is transitive when everyone must always agree is no longer transitive, and the known techniques do not extend directly.

Whence from here? The Sequenced Stopping-Fault model may have useful applications, and should be investigated further. The problem of choosing an optimal broadcast sequence for this or other interesting problems appears hard, however.

Proofs of the lower bounds might provide interesting tools, and constructive disproofs would provide faster protocols. Both would increase our understanding of "what we know that he knows that you know that they know about what we all know!"

5. References

[BT81] Bermond, J.C., and Thomassen, C., "Cycles in Digraphs--A Survey," *Journal of Graph Theory*, Vol. 5 (1981) 1-43.

[CS83] Chvatal, V. and Szemeredi, E., "Short Cycles in Directed Graphs," Technical Report SOCS-82.11, 1983.

[DLM82] DeMillo, R., Lynch, N., and Merritt, M. "Cryptographic Protocols" *Proc. 14th ACM Symposium on the Theory of Computing*, 1982.

[DS82] Dolev, D., and Strong, H. R. "Polynomial Algorithms for Multiple Processor Agreement." *Proc. 14th ACM Symposium on the Theory of Computing*, 1982, pp. 401-407.

[D84] Dwork, C. private communication.

[FL82] Fischer, M.J., and Lynch, N.A. "A Lower Bound on the Time to Assure Interactive Consistency." *Information Processing Letters* 14, 4 (1982), 183-186.

[LF84] Lamport, L. and Fischer, M.J. "Byzantine Generals and Transaction Commit Protocols." Manuscript.

[PSL80] Pease, M., Shostak, R., and Lamport, L. "Reaching Agreement in the Presence of Faults." *Journal of the ACM* 27, 2 (1980), 228-234.