Yale University
Department of Computer Science

The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)

Michael J. Fischer

YALEU/DCS/TR-273
June 1983
Reissued February 2000

To be presented at the International Conference on Foundations of Computation Theory, Borgholm, Sweden, August 21–27, 1983.

[This is a reissue of Research Report YALEU/DCS/RR-273, June 1983, reformatted in LaTeX. It also appeared in M. Karpinsky, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 127–140, Spring-Verlag, 1983.]

The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)*

Michael J. Fischer
Yale University
New Haven, Connecticut

Abstract

Agreement problems involve a system of processes, some of which may be faulty. A fundamental problem of fault-tolerant distributed computing is for the reliable processes to reach a consensus. We survey the considerable literature on this problem that has developed over the past few years and give an informal overview of the major theoretical results in the area.

1 Agreement Problems

To achieve reliability in distributed systems, protocols are needed which enable the system as a whole to continue to function despite the failure of a limited number of components. These protocols, as well as many other distributed computing problems, requires cooperation among the processes. Fundamental to such cooperation is the problem of agreeing on a piece of data upon which the computation depends. For example, the data managers in a distributed database system need to agree on whether to commit or abort a given transaction [20, 26]. In a replicated file system, the nodes might need to agree on where the file copies are supposed to reside [19, 30]. In a flight control system for an airplane [35], the engine control module and the flight surface control module need to agree on whether to continue or abort a landing in progress. The key point here is not *what* the processes are agreeing on but the fact that they must all come to the *same* conclusion.

An obvious approach to achieving agreement is for the processes to vote and agree on the majority value. In the absence of faults, this works fine, but in a close election, the vote of one faulty process can swing the outcome. Since distinct

*This work was supported in part by the Office of Naval Research under Contract N00014–82–K–0154, and by the National Science Foundation under Grant MCS–8116678.

reliable processes might receive conflicting votes from a faulty process, they might also reach conflicting conclusions about the outcome of the election and hence fail to reach agreement. Davies and Wakerly [2] realized this difficulty and proposed a multistage voting scheme to overcome the problem.

A simple form of the problem is to achieve consensus on a single bit. Assume a fixed number of processors, some of which are initially faulty or may fail during the execution of the protocol. Each processor i has an initial bit x_i. The *consensus* problem is for the non-faulty processes to agree on a bit y, called the *consensus value*. More precisely, we want a protocol such that each reliable process i eventually terminates with a bit y_i, and $y_i = y$ for all i.

y in general will depend in some way on the initial bits x_i. In the absence of such a requirement, the problem becomes trivial, for each process can simply choose $y_i = 0$. Some dependency requirements that have been studied, in order of increasing strength, are:

1. (non-triviality): For each $y \in \{0, 1\}$, there is some initial vector x_i and some admissible execution of the protocol in which y is the consensus value. (The qualification "admissible" allows for additional restrictions, such as bounds on the number of faulty processes, on the kinds of computations we are willing to consider.)

2. (weak unanimity): If $x_i = x \in \{0, 1\}$ for all i, then $y = x$, provided that no failures actually occur during the execution of the protocol.

3. (strong unanimity): If $x_i = x \in \{0, 1\}$ for all i, then $y = x$.

Two other closely related problems have been studied extensively in the literature. The *interactive consistency* problem is like the consensus problem except that the goal of the protocol is for the non-faulty processes to agree on a vector $\mathbf{y}$, called the *consensus vector*. Again, we add dependency requirements:

1. (weak): for each j, $\mathbf{y}_j = x_j$ if j is non-faulty, provided that no failures actually occur during the execution of the protocol.

2. (strong): for each j, $\mathbf{y}_j = x_j$ if j is non-faulty.

Finally, in the *generals* problem or *reliable broadcast* problem, one assumes a distinguished processor (the "general" or "transmitter") which is trying to send its initial bit x to all the others. As before, all the reliable processes have to reach consensus on a bit, and we add dependency requirements:

1. (weak): $y = x$ if no failures occur during the execution of the protocol.

2. (strong): $y = x$ if the general is non-faulty.

Without further qualification, any reference to one of these problems will refer to the version with the strong dependency requirement.

2 Models of Computation

The kinds of solutions that can be obtained to agreement problems depend heavily on the assumptions made about the model of computation and the kinds of faults to which it is prone. Throughout this paper, we will assume a fixed number n of processes. A protocol is said to be *t-resilient* if it operates correctly as long as no more than t processes fail before or during execution.

We consider two kinds of processor faults. A *crash* occurs when a process stops all activity. Up to the point of the crash, it operates correctly and after that it is completely inactive. A protocol that can tolerate up to t crashed processes is said to be *t-crash resilient*. We do not concern ourselves with the problem of repairing a faulty process and reintegrating it into the system, although that of course is a crucial problem in the practical implementation of any of these ideas [28].

A more disruptive kind of failure is the so-called *Byzantine failure* [1] in which no assumptions are made about the behavior of a faulty process. In particular, it can send messages when it is not supposed to, make conflicting claims to other processes, act dead for awhile and then revive itself, etc. A protocol that can tolerate up to t processes which exhibit Byzantine failures is said to be *t-Byzantine resilient* and is sometimes called a *Byzantine* protocol. The problem of finding a t-Byzantine resilient protocol for the (weak) generals problem is called the *(weak) Byzantine generals* problem.

To show that a protocol is Byzantine resilient, one has to consider all possible faulty behaviors, including those in which the failed processes act maliciously against the protocol. This doesn't mean that Byzantine protocols are only appropriate in adversary situations. The folklore is full of stories in which systems failed in bizarre and unexpected ways, and in the absence of good ways of characterizing the kinds of failures that occur in practice, protecting against Byzantine failures is a conservative approach to reliable systems design.

We assume the message system to be completely reliable and that only processes are subject to failure. We also assume that any process can reliably determine the sender of any message it receives, and any message so delivered arrives intact and without errors. Unless stated otherwise, we assume the network is a completely connected graph.

[1] The terminology comes from [25], in which a fable is recounted concerning a problem of military communications in times of old.

Of course, in real systems, communication links as well as processors are subject to failure. However, a link failure can be identified with the failure of one of the processors at its two ends, so a t-resilient protocol automatically tolerates up to t process and link failures. Nevertheless, this may give an overly pessimistic view of the reliability of the system. Reischuk [32] greatly refines the fault assumptions, enabling him to obtain more informative results on the actual behaviors of the systems.

A crucial assumption concerns whether or not the failure of a process to send an expected message can be detected. If so, then the expectant receiver gains the valuable knowledge that the sender is faulty. In a model with accurate clocks and bounds on message transit times, such detection is possible through the use of timeouts. (Cf. [21].) Also, detection is automatic in a synchronous model in which the processes run in lock step and messages sent at one step are received at the next. However, detection is impossible in a fully asynchronous model in which no assumptions are made about relative step times or message delays, for there is no way to tell whether the sender has failed or is just running very slowly. This turns out to have a profound effect on the solvability of agreement problems.

We use the terms *synchronous* and *asynchronous* to distinguish between these two extreme cases, while remaining fully cognizant of the fact that synchronous message behavior can be achieved in systems with weaker assumptions than full synchrony. For our purposes, we will assume that a synchronous computation proceeds in a sequence of *rounds*. In each round, every process first sends as many messages as it wishes to other processes, and then it receives the messages sent to it by other processes. Thus, messages received during a round cannot affect the messages sent during the same round.

One further significant assumption is whether or not the model supports signatures. We assume that the author of a signed message can be reliably determined by anyone holding the message, regardless of where the message came from and regardless of anything that the faulty processes might have done. In other words, signatures cannot be forged by faulty processes, so if C receives a message from B signed by A, then C knows that A really sent the message and that it was not fabricated by B. Signatures, too, have a profound effect on the solvability of agreement problems. We sometimes use *authenticated* to refer to a protocol using signed messages.

Digital signatures can be implemented using cryptographic techniques [3, 4, 27, 33], or if one is willing to assume that faulty processes are not malevolent, simple signature schemes which are not cryptographically secure can be used instead. All that we require is that it be unlikely for a faulty process to generate a valid signature of some other process. Note that no special techniques are needed to implement signatures if only crashes (and not Byzantine failures) are considered, for then no incorrect messages are ever sent.

The practicality of agreement protocols depends heavily on their computational complexity. Some factors that might be important are the amount of *time* needed to complete the protocol, the amount of *message traffic* generated, or the amount of *memory* needed by the participants. All of these quantities are in general dependent on which faults actually occur and when. A reasonable assumption in many situations is that faults happen rarely, so it is acceptable to spend considerable resources handling them, but one wants the normal case to be handled quite efficiently. Note however that in a very large system, the probability of at least one fault is high, and the expected number of faults grows linearly with the size of the system.

We measure time in terms of the number of rounds of message exchange that take place. Thus, we assume every process can potentially exchange messages with every other in a single unit of time. Just how realistic this notion of time is depends highly on the structure of the message system and on the reasonableness of the assumption that a process can really send or receive n messages in a single time step. We measure message traffic variously as the total number of messages sent, the total number of bits in those messages, or the number of signatures (in the case of an authenticated protocol).

3 Relations Among Agreement Problems

The three agreement problems are closely related. The generals problem is a special case of the interactive consistency problem in which only one process's initial value is of interest, so a protocol achieving interactive consistency also solves the generals problem. Conversely, n copies of a protocol for the generals problem can be run in parallel to solve the interactive consistency problem.

The consensus problem appears to be slightly weaker than the other two. An interactive consistency algorithm can be modified to solve the consensus problem by just having each process choose as its consensus value the majority value in the consensus vector. This works as long as fewer than 1/2 of the processes are faulty.

Using a consensus algorithm to solve either of the other two problems, however, seems to require an additional round of information exchange. For example, the general's problem can be solved as follows:

Algorithm I

1. The general sends its value to each of the other processes.

2. All of the processes together run a consensus algorithm using as initial values the bits received from the general at the first step. (The general of course uses

its own bit.)

This solves the generals problem since if the general is reliable, then all of the processes receive the same value in step 1. By the strong unanimity condition, this value will be chosen as the caonsensus value. In any case, agreement is reached. The extra cost is one additional round of 1-bit messages in step 1. Thus, we have proved:

Theorem 1 *Given a t-resilient solution to the consensus problem, there is a t-resilient solution to the generals problem which uses one more "round" of message exchange and sends $n-1$ additional messages of 1-bit each.*

Many solutions to the generals problem have the general structure of Algorithm I and thus appear to have embedded within them solutions to the consensus problem, seemingly obviating the need for Algorithm I and the extra round of messages. However, the embedded consensus algorithm does not necessarily solve the full consensus problem, for the case in which the general is reliable yet the x_i's are not all the same can never arise when the x_i's are obtained from the general on the first step.

Similar remarks apply to the corresponding weak versions of these problems. In fact, a weak Byzantine generals algorithm solves the weak consistency problem directly (without first using it to solve the interactive consistency problem), for if all the initial values are the same and no process is faulty, then it suffices to simply agree on the general's value. There is not, however, any readily apparent way to use a solution to any of the weak versions of the agreement problem to solve any of the strong ones. In fact, for a slightly different "approximate" agreement problem, Lamport [22] shows that the weak version has a solution whereas the strong one does not.

4 Solvability of Agreement Problems

Perhaps the most basic question to ask of a proposed agreement problem is whether or not it has a solution at all. By the previous discussion and Theorem 1 the consensus problem and the interactive consistency problem have t-resilient solutions iff the generals problem does, so we will restrict attention to the latter problem in this section.

Consider first the synchronous case. With signatures, Pease, Shostak, and Lamport [25, 29] give a t-resilient solution for any t.

Theorem 2 *There is a t-resilient authenticated synchronous protocol which solves the strong (weak) Byzantine generals problem.*

Briefly, the protocol consists of $t+1$ rounds. In the first round, the general sends a signed message with its value to each other process. At each round thereafter, each process adds its signature to each valid message received from the previous round and sends it to all processes whose signature does not already appear on the message. A message received during round k is *valid* if it bears exactly k distinct signatures, the first of which is the general's. Let V_i be the set of values contained in all the valid messages received by i through the end of round $t+1$. If V_i is a singleton, then that value is chosen as the consensus value. Otherwise, a fixed constant NIL is chosen.

To prove agreement, we argue that if i and j are both reliable, then $V_i = V_j$. There are two cases to consider. If the general is reliable, then both V_i and V_j consist solely of the general's value, since no other value ever appears in a valid message. Otherwise, consider the message M from which i first learned of v. M consists of v followed by a list of distinct signatures $m_1, \ldots, m_k$, the first of which is the general's, and $k \leq t+1$. If $k < t+1$ and process j does not already know about v, then j learns of v from i on the next round. If $k = t+1$, then $m_1, \ldots, m_t$ are all faulty or else i would have learned of v earlier. But then m_{t+1} is reliable, so j learns of v at the same round as i. Correctness of the protocol easily follows.

Without signatures, there is a solution if and only if the fraction of faulty processes is not too large.

Theorem 3 *There is a t-resilient synchronous protocol without authentication which solves the strong (weak) Byzantine generals problem iff $t/n < 1/3$.*

The impossibility argument for $t/n \geq 1/3$ appears in [25, 29] for the strong case and in [22] for the weak case of the problem. Protocols demonstrating the solvability of both problems for $t/n < 1/3$ appear in [25, 29]. Various protocols have since appeared with additional desirable properties, some of which will be discussed later in this paper.

In the fully asynchronous case, there is no solution. In fact, Fischer, Lynch, and Paterson [18] show that the problem remains unsolvable even with much weaker requirements:

Theorem 4 *In a fully asynchronous environment, there is no 1-crash resilient solution to the consensus problem, even when only the non-triviality condition is required.*

The proof is by contradiction. In general outline, one assumes the existence of such a protocol. The protocol is *committed* to the eventual consensus value at a certain point in time if thereafter only the one value is a possible outcome, no matter how processes are scheduled or how messages are delivered. One shows that at least for some initial configuration, the outcome is not already committed. Starting from there, one constructs an infinite computation such that the system forever stays uncommitted, contradicting the assumed correctness of the protocol. The details get somewhat involved since it is necessary to insure that the infinite computation results from a "fair" schedule. The interested reader is referred to [18].

Returning to the Byzantine generals problem in a synchronous environment, we consider weaker connectivity assumptions on the network which nonetheless permit a solution. With signatures, Lamport et al. [25] show that the Byzantine Generals problem can be solved in any network in which the reliable processes are connected. Without signatures, they show that a solution is possible in a $3t$-"regular" graph. Dolev [5, 6] extends this latter result to completely characterize the networks in which the problem is solvable:

Theorem 5 *Consider a synchronous network with connectivity k having n processors, t of which may be faulty. Then the Byzantine generals problem is solvable without authentication iff $t/n < 1/3$ and $t/k < 1/2$.*

Three recent unpublished results deserve brief mention, all of which extend the asynchronous model slightly in order to avoid the assumptions of Theorem 4. Ben-Or [1] allows randomized algorithms and shows that crash-resilient consensus is achievable with probability 1 when $t/n < 1/2$, and Byzantine-resilient consensus is achievable with probability 1 when $t/n < 1/5$. Rabin [31] uses randomized algorithms with an initial random "deal" and signatures to achieve certain agreement with an expected number of rounds that is only 4, independent of n and t, so long as $t/n < 1/4$. Finally, Dolev, Dwork, and Stockmeyer [7] distinguish among the different kinds of asynchrony in the model of [18] to get tighter conditions on when consensus protocols are and are not possible.

5 Complexity Results

5.1 Upper Bounds

The t-resilient Byzantine generals algorithms of [25, 29] take time $t + 1$ and send a number of message bits that is exponential in t. The first algorithm to use only a polynomial number of message bits was found by Dolev and Strong [12] and subsequently improved by Fischer, Fowler, and Lynch [16]. The still stronger result below is from [8].

Theorem 6 *Let $t/n < 1/3$. There is a t-resilient solution without authentication to the Byzantine generals problem which uses $2t + 3$ rounds of information exchange and $O(nt + t^3 \log t)$ message bits.*

It remains an open problem if there is any unauthenticated algorithm which simultaneously achieves fewer than $2t + 3$ rounds and uses only polynomially many message bits.

With authentication, and counting number of messages instead of message bits, we get:

Theorem 7

(a) *There is a t-resilient authenticated solution to the Byzantine generals problem which uses $t + 1$ rounds and sends $O(nt)$ messages;*

(b) *There is a t-resilient authenticated solution to the Byzantine generals problem which uses $O(t)$ rounds and sends only $O(n + t^2)$ messages.*

Part (a) was shown by Dolev and Strong [15], and part (b) was shown by Dolev and Reischuk [10].

For practical applications, these bounds are not very encouraging, especially the $t + 1$ bound on the number of rounds. As we shall see, this bound cannot be improved in the worst case that t faults actually occur. However, Dolev, Reischuk and Strong [11, 14] have looked at the question of whether Byzantine generals solutions exist which stop early when fewer faults occur. The answer depends on whether synchronization upon termination is also required.

For definiteness, we say that a process *halts within* r rounds if it is non-faulty and it chooses its consensus value and enters a stopping state before sending or receiving any round $r + 1$ messages. It *halts in* round r if it halts within r rounds but does not halt within $r - 1$ rounds. An agreement protocol *terminates* when all reliable processes have halted. If it terminates, we say it reaches *immediate* agreement if all reliable processes halt in the same round, and it reaches *eventual* agreement otherwise. Thus, immediate agreement serves to synchronize the processes as well as enabling them to agree on a value. Note that all of the protocols discussed previously achieve immediate agreement since all processes choose their consensus value in the last round.

The following theorem is from [11]:

Theorem 8 *Let $t/n < 1/3$. There is a t-resilient protocol without authentication which solves the Byzantine generals problem and reaches eventual agreement within* $\min(2t + 3, 2f + 5)$ *rounds, where $f \leq t$ is the actual number of faults.*

The same paper also contains a more refined protocol which stops even earlier when t is only about $\sqrt{n}$.

If one assumes processes can fail only by crashing, then Lamport and Fischer show that these bounds can be improved [23].

Theorem 9 *There is a t-crash resilient protocol (without authentication) which solves the generals problem and reaches eventual agreement by the end of round $f + 2$, where $f \leq t$ is the actual number of crashes.*

We give the protocol and sketch its proof. There are only four possible messages — 0, 1, NIL, and ϕ. 0, 1 are the two possible initial values of the general, ϕ means "I don't know", and NIL is a default consensus value which is chosen when crashes prevent the reliable processes from discovering the general's value.

Algorithm II

A. Round 1: Process 1 (the general) sends its value to every process.

B. Round r, $\mathbf{1 < r \leq t + 1}$: Each process does the following:

1. If it received a value $v \in \{0, 1, \mathrm{NIL}\}$ from any process in round $r - 1$, then it:
   - takes v as its consensus value;
   - sends v to every process;
   - halts.
2. Otherwise, if it received ϕ during round $r - 1$ from every process not known to have crashed before the beginning of that round, then it:
   - takes NIL as its consensus value;
   - sends NIL to every process;
   - halts.

   (It knows a process has crashed if it failed to receive an expected message from it during the previous round.)
3. Otherwise, it sends ϕ to every process.

C. End of Round $\mathbf{t + 1}$: Each process that has not halted does the following:

1. If it received a value $v \in \{0, 1, \mathrm{NIL}\}$ from any process during round $t + 1$, then it takes v as its consensus value and halts.
2. Otherwise, it chooses NIL as its consensus value and halts.

Correctness of the algorithm follows readily from the following facts. Recall that a crashed process is not considered to be halted.

1. If some process halts at step B1 or B2 during round r and chooses value v, then every other process which halts at step B1 or B2 during round r also chooses v.

2. If some process halts at step B1 or B2 during round r and chooses value v, then every reliable process which has not already halted will choose v and halt at step B1 in round $r+1$ (if $r < t+1$) or at step C1 in round $t+1$ (if $r = t+1$).

3. If no process crashes or halts during round $r > 1$, then ϕ is the only message sent during that round.

4. If any process terminates at step C2 in round $t+1$, then all reliable processes do.

Moreover, if fewer than k processes crash in the first k rounds, then the protocol terminates within $k+1$ rounds; hence if there are at most f crashes, then the protocol terminates within $f+2$ rounds.

A more elaborate protocol with similar abstract properties but which is quite possibly more efficient in practice appears in [34].

5.2 Lower Bounds

All of the protocols above use $t+1$ rounds in the worst case. Fischer and Lynch [17] present a proof that $t+1$ rounds are necessary for achieving interactive consistency without signatures and hence also for solving the unauthenticated Byzantine generals problem. Several people have extended this result in one way or another. DeMillo, Lynch, and Merritt [3, 27] and independently Dolev and Strong [12, 15] show that the t+1 lower bound holds for authenticated solutions to the Byzantine generals problem. Lamport and Fischer [23], by a similar proof, show that the same bound holds assuming that the protocol is only crash resilient and solves the weak consensus problem, but they did not consider the authenticated case. We summarize these results below.

Theorem 10 *Assume $t \leq n-2$.*

(a) Every t-resilient protocol without signatures for the weak consensus problem uses at least $t+1$ rounds of message exchange in the worst case.

(b) Every t-resilient authenticated protocol for the Byzantine generals problem uses at least $t+1$ rounds of message exchange in the worst case.

We note that the weak consensus problem has not been explicitly studied with signed messages, but we conjecture that the same bound will still hold.

We sketch the basic structure underlying these proofs, although much more is involved in really making them go through. For two distinct computations S and T, define $S \sim T$ if S and T "look" the same to some reliable process p, that is, p receives the same messages and behaves exactly the same in both S and T. Hence, p chooses the same consensus value in each, which must be the consensus value for both S and T. Now, the proof proceeds by assuming at most t rounds and then constructing a sequence of t-round computations S_0, $S_1, \ldots, S_k$ such that S_0 has consensus value 0, S_k has consensus value 1, and $S_{i-1} \sim S_i$ for $1 \leq i \leq k$. This results in a contradiction. The constructions need one faulty process per round; hence, they cannot be used to find computations of more than t rounds.

Dolev and Strong [14] show that $t+1$a rounds are needed in a t-resilient immediate Byzantine generals protocol even when the actual number of failures is less. These theorems also appear without proofs in [11].

Theorem 11 *Let $t \leq n - 2$, and let P be a t-resilient (authenticated) protocol solving the Byzantine generals problem which always reaches immediate agreement. Then it is possible for P to run for at least* t + 1 *rounds even when there are no faults.*

In the case of eventual agreement, they prove the following:

Theorem 12 *Let P be a t-resilient (authenticated) protocol solving the Byzantine generals problem which reaches eventual agreement, and let $f < t$. Then it is possible for P to run for at least $f + 2$ rounds with only f faults.*

We conjecture that this can be extended to t-crash resiliant generals protocols, which would then show the optimality of 9.

Finally, we look at lower bounds on the number of messages and signatures needed. Dolev and Reischuk [10] show:

Theorem 13 *The total number of messages and signatures in any t-resilient (authenticated) Byzantine generals solution is $\Omega(nt)$.*

Theorem 6 shows that this bound is tight when n is large relative to t. If one counts only messages, then they show

Theorem 14 *The total number of messages in any t-resilient (authenticated) Byzantine generals solutions is $\Omega(n + t^2)$.*

Theorem 7, part (b) shows this bound "best possible" for authenticated algorithms.

6 Applications of Agreement Protocols

The abstract versions of agreement problems considered in this survey are not general enough to be directly applicable to many practical situations. We mention here some extensions and applications of these problems.

First of all, one often wants to reach agreement on a value from a larger domain than just $\{0, 1\}$. If the domain has v elements, then one can encode the elements in binary and run $\lceil \log_2 v \rceil$ copies of the agreement protocol, one for each bit, but more efficient algorithms might be possible. In applications such as clock synchronization, the domain of values can be taken to be the real numbers, and only approximate agreement is needed. Lamport and Melliar-Smith [24] studies the clock synchronization problem, and Dolev, Lynch, and Pinter [9] look at the abstract approximate agreement problem.

A difficult part of implementing these algorithms is building message systems which actually have the reliability and synchronization properties that were assumed in the models. Real distributed systems are quasi-asynchronous, and to avoid the difficulties of Theorem 4 one must make reasonable timing assumptions and make effective use of clocks and timeouts. Lamport [21] gives some insights as to how this can be done.

Finally, we should mention the papers by Dolev and Strong [13] and Mohan, Strong, and Finkelstein [28] that describe serious attempts to apply agreement protocols to real problems of distributed databases.

7 Acknowledgement

The author is grateful for Ming Kao for help in assembling the bibliography and to Paul Hudak for many helpful comments on an early draft of this paper.

References

[1] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, 1983. To appear.

[2] D. Davies and J. F. Wakerly. Synchronization and matching in redundant systems. *IEEE Transactions on Computers*, C-27(6):531–539, June 1978.

[3] R. A. DeMillo, N. A. Lynch, and M. J. Merritt. Cryptographic protocols. In *Proc. 14th ACM Symposium on Theory of Computing*, pages 383–400, 1982.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22:644–654, 1976.

[5] D. Dolev. Unanimity in an unknown and unreliable environment. In *Proc. 22nd IEEE Symposium on Foundations of Computer Science*, pages 159–168, 1981.

[6] D. Dolev. The Byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982.

[7] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. Manuscript, 1983.

[8] D. Dolev, M. J. Fischer, R. Fower, N. A. Lynch, and H. R. Strong. An efficient Byzantine agreement without authentication. *Information and Control*, to appear. See also IBM Research Report RJ3428 (1982).

[9] D. Dolev, N. A. Lynch, and S. Pinter. Reaching approximate agreement in the presence of faults. Manuscript, 1982.

[10] D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine agreement. In *Proc. ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 132–140, 1982.

[11] D. Dolev, R. Reischuk, and H. R. Strong. 'Eventual' is earlier than 'immediate'. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 196–203, 1982.

[12] D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agremment. In *Proc. 14th ACM Symposium on Theory of Computing*, pages 401–407, 1982.

[13] D. Dolev and H. R. Strong. Distributed commit with bounded waiting. In *Proc. Second Symposium on Reliability in Distributed Software and Database System*, Pittsburgh, July 1982.

[14] D. Dolev and H. R. Strong. Requirements for agreement in a distributed system. In *Proc. Second International Symposium on Distributed Data Bases*, Berlin, September 1982.

[15] D. Dolev and H. R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.*, to appear. See also IBM Research Report RJ3416 (1982).

[16] M. J. Fischer, R. J. Fowler, and N. A. Lynch. A simple and efficient Byzantine generals algorithm. In *Proc. Second IEEE Symposium on Reliability in Distributed Software and Database Systems*, pages 46–52, Pittsburgh, 1982.

[17] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982.

[18] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. In *Proc. Second ACM Symposium on Principles of Database Systems*, March 1983.

[19] D. K. Gifford. Weighted voting for replicated data. Technical Report CSL-79-14, XEROX Palo Alto Reserach Center, September 1979.

[20] J. Gray. A discussion of distributed systems. Research Report RJ2699, IBM, September 1979.

[21] L. Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, to appear. See also technical report, Computer Science Laboratory, SRI International (June 1981).

[22] L. Lamport. The weak Byzantine generals problem. *Journal of the ACM*, 30(3), July 1983. To appear.

[23] L. Lamport and M. J. Fischer. Byzantine generals and transaction commit protocols. Manuscript, 1982.

[24] L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. Technical report, Computer Science Laboratory, SRI International, March 1982.

[25] L. Lamport, R.. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[26] B. G. Lindsay et al. Notes on distributed databases. Research Report RJ2571, IBM, July 1979.

[27] M. J. Merritt. Cryptographic protocols. Technical Report GIT-ICS-83/06, School of Inf. & Comp. Sci., Georgia Institute of Techonology, February 1983.

[28] C. Mohan, H. R. Strong, and S. Finkelstein. Method for distributed transaction commit and recovery using Byzantine agreement within clusters of processors. Research Report RJ3882, IBM, 1983.

[29] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[30] G. Popek et al. LOCUS: A network transparent, high reliability distributed system. In *Proc. 8th ACM Symposium on Operating Systems Principles*, pages 169–177, December 1981.

[31] M. Rabin. Randomized Byzantine generals. Manuscript, 1983.

[32] R. Reischuk. A new solution for the Byzantine generals problem. Research Report RJ3673, IBM, November 1982.

[33] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[34] F. B. Schneider, D. Gries, and R. D. Schlichting. Fast reliable broadcasts. Computer Science Technical Report TR 82-519, Cornell University, September 1982.

[35] J. H. Wensley et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proc IEEE*, 66(10):1240–1255, October 1978.