Asynchronous Consensus and Broadcast Protocols

GABRIEL BRACHA AND SAM TOUEG

Cornell University, Ithaca, New York

Abstract. A consensus protocol enables a system of n asynchronous processes, some of which are faulty, to reach agreement. There are two kinds of faulty processes: fail-stop processes that can only die and malicious processes that can also send false messages. The class of asynchronous systems with fair schedulers is defined, and consensus protocols that terminate with probability 1 for these systems are investigated. With fail-stop processes, it is shown that $\lceil (n + 1)/2 \rceil$ correct processes are necessary and sufficient to reach agreement. In the malicious case, it is shown that $\lceil (2n + 1)/3 \rceil$ correct processes are necessary and sufficient to reach agreement. This is contrasted with an earlier result, stating that there is no consensus protocol for the fail-stop case that always terminates within a bounded number of steps, even if only one process can fail. The possibility of reliable broadcast (Byzantine Agreement) in asynchronous systems is also investigated. Asynchronous Byzantine Agreement is defined, and it is shown that $\lceil (2n + 1)/3 \rceil$ correct processes are necessary and sufficient to achieve it.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network, Protocols—*protocol architecture*, C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications; distributed databases; network operating systems*; C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism*; H.2.4 [**Database Management**]: Systems—*distributed systems; transaction processing*

General Terms: Algorithms, Reliability, Probability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, consensus problem, distributed computing, fault tolerance, impossibility proof, probabilistic algorithms, reliability

1. *Introduction*

In this paper we consider protocols for reaching agreement in an unreliable asynchronous distributed system. Numerous variations of this question appeared in the literature. These differ in the assumed message system properties, the kind of failures accorded to the processes, and the notion of what constitutes a solution.

In our model all the processes are fully interconnected. The message system is reliable, though completely asynchronous, so messages can be delayed arbitrarily long. We consider two types of faulty processes. Fail-stop processes [9] may simply "die," that is, stop participating in the protocol. However, there is no way to detect the death of such a process, and distinguish between a dead process and a merely slow one. Malicious processes [6], beside failing to send the required messages, may also send false and contradictory messages, even according to some malevolent plan.

Each process starts with some initial value. At the conclusion of the protocol all the correct processes must agree on the same value. However, we are ruling out

Authors' address: Department of Computer Science, Cornell University, 405 Upson Hall, Ithaca, NY 14853.

the trivial case that the agreed value is fixed regardless of the processes' initial input.

This type of system (with fail-stop processors) was investigated in [3]. Fischer et al. showed the impossibility of a consensus protocol if only one failure may occur. However, in [3], the notion of an admissible solution is a protocol that always terminates within a finite number of steps. In this paper we are interested in a different kind of solution: we consider protocols that may never terminate, but this would occur with probability 0, and the expected termination time is finite. There are two ways to introduce probabilities on the possible executions of a protocol. In the first approach [1], random steps are introduced in the protocol. The other approach, and the one we adopt in this paper, is to postulate some probabilistic behavior about the message system.

In the case of fail-stop processors, we describe a probabilistic protocol that can withstand up to $\lfloor(n-1)/2\rfloor$ failures, where n is the number of processes. We also show that there is no consensus protocol that can overcome more than $\lfloor(n-1)/2\rfloor$ failures. With malicious processes, we describe a protocol that can withstand up to $\lfloor(n-1)/3\rfloor$ failures. We also prove the impossibility of such a protocol if more than $\lfloor(n-1)/3\rfloor$ processes may fail.

2. *The Fail-Stop Case*

2.1 THE MODEL. We consider an asynchronous system of n fully interconnected processes. Processes communicate by sending *messages* via the *message system*. The message system maintains for each process a *message buffer* of messages sent but not yet received. It also supports the following primitives for each process q.

send(p, m). Instantaneously place the message m in process p's buffer.

receive(m). Remove some message from q's buffer and return it in m, or return the null value $\emptyset$. This choice is made nondeterministically. (Returning $\emptyset$, even if the buffer is not empty, is a device to model the arbitrarily long transmission delays incurred in a message system.)

Each process has an unbounded internal storage whose value constitutes its *state*. In an *atomic step* of the system, a process can try to receive a message, perform an arbitrarily long local computation, and then send a finite set of messages. The computation and the messages sent are prescribed by the *protocol*, that is, a function of the message received and the local state.

A *correct* process always follows the protocol until the protocol completion. A *fail-stop* process may *die* during the execution of the protocol, that is, it may stop participating in the protocol. The death of a process occurs without warning messages. From our model, it is clear that such a death can not be detected by other processes. In particular, there is no way to distinguish between a dead process and a merely slow one.

Each process p has two distinguished memory locations, *input* i_p and *decision* d_p. The system starts with all the processes in some initial state, all the buffers empty, d_p undefined, and i_p having some value in $\{0, 1\}$. The protocol can assign to d_p a value in $\{0, 1\}$. Once d_p is assigned a value v, it cannot be changed, and p is said to have *decided* v.

The *configuration* of the system is the collection of the states and the buffers' contents of all the processes. Let C be a configuration and S be a subset of processes. A *subconfiguration* C_S is the restriction of C to the members of S. Henceforth, we reserve the notation F^i to denote any configuration where all the correct processes have decided i, and F^i_S to denote any subconfiguration where all the correct processes in S have decided i.

A sequence of atomic steps is called a *schedule*. If the execution of a schedule σ from a configuration C results in a configuration D, we write $C \overset{\sigma}{\vdash} D$. If there is a schedule σ such that $C \overset{\sigma}{\vdash} D$, we write $C \vdash D$; if all the processes performing atomic steps in σ belong to a subset of processes S, then we write $C_S \vdash D_S$, and say that D_S is *reachable from* C_S. The configuration D is said to be *reachable* if it is reachable from some initial configuration.

We postulate an agent, the *scheduler*, that will determine the next atomic step in the execution. Probabilistic assumptions on the behavior of the scheduler provide us with a probability measure on the space of all possible schedules. In this paper we are interested in the class of *fair schedulers*, as defined in Section 2.3.

A *k-resilient consensus protocol* is a protocol that satisfies the following properties, provided that no more than k processes are faulty:

(1) *Bivalence.* If all the processes are correct, both F^0 and F^1 configurations are reachable.
(2) *Consistency.* There is no reachable configuration where correct processes decide different values.
(3) *Convergence.* For any initial configuration, $\lim_{t\to\infty} \Pr[$a correct process has not decided within t steps$] = 0$.

Note that, from the consistency and the convergence properties of k-resilient protocols, if $C_S \vdash F^i_S$, then $C \vdash F^i$.

2.2 A LOWER BOUND ON THE NUMBER OF CORRECT PROCESSES. The possibility of undetectable deaths during the execution of the protocol implies that, at any stage of the protocol, processes will have to act relying only on partial information about the state of the system. This is formalized by the following lemma.

LEMMA 1. *With a k-resilient consensus protocol, for any reachable configuration C, and for any subset S of processes that contains at least n − k correct processes, either* $C_S \vdash F^0_S$ *or* $C_S \vdash F^1_S$.

PROOF. Let C be a reachable configuration, S be a subset of processes that contains at least $n - k$ correct processes, and $\bar{S}$ be the complement of S (i.e., the set of processes that are not in S). Note that $|\bar{S}| \le k$. Assume first that all the processes in $\bar{S}$ are fail-stop. Suppose that, after reaching configuration C, all the processes in $\bar{S}$ die without sending warning messages. This results in a configuration C'. We have $C'_S = C_S$. From the consistency and the convergence properties of the k-resilient protocol, we must have $C'_S \vdash F^0_S$ or $C'_S \vdash F^1_S$. Since $C'_S = C_S$, and since the death of processes in $\bar{S}$ cannot be detected, we have $C_S \vdash F^0_S$ or $C_S \vdash F^1_S$. This must also hold even if there are correct processes in $\bar{S}$. □

Let C be a configuration, and S a subset of processes as in Lemma 1. If both $C_S \vdash F^0_S$ and $C_S \vdash F^1_S$, then C_S is *bivalent*. If $C_S \vdash F^0_S$ or $C_S \vdash F^1_S$, but not both, then C_S is *univalent* (0-*valent* or 1-*valent*, accordingly).

LEMMA 2 [3]. *For* $k \ge 1$, *any k-resilient consensus protocol has a bivalent initial configuration.*

PROOF. Suppose all the processes are correct. Initial configurations differ only by the processes' input values. Two initial configurations differing by the input value of only one process are *adjacent.* Assume, for contradiction, there is a k-resilient protocol such that any initial configuration is either 0-valent or 1-valent. By the bivalence property of the protocol, there must be one of each. Therefore, there must be two adjacent initial configurations, I^0 and I^1, that are 0-valent and 1-valent, respectively. These configurations differ only by the input value of some process p. Therefore, $I_S = I_S$, where S includes all the processes except p. From Lemma 1, either $I^0_S \vdash F^0_S$ or $I^0_S \vdash F^1_S$. If $I^0_S \vdash F^0_S$, then $I^1_S \vdash F^0_S$, and therefore we have $I^1 \vdash F^0$. But I^1 is 1-valent, a contradiction. A similar contradiction is obtained if we assume $I^0_S \vdash F^1_S$. □

THEOREM 1. *There is no* $\lceil n/2 \rceil$*-resilient consensus protocol for the fail-stop case.*

PROOF. Assume there is such a protocol, and consider a system in which all the processes are correct. Let C be any reachable configuration, S be any subset of processes of size $\lfloor n/2 \rfloor$, and $\bar{S}$ be the complement of S. We claim that C_S and $C_{\bar{S}}$ are either both 0-valent or both 1-valent.

From Lemma 1, since the protocol is $\lceil n/2 \rceil$-resilient and $|S|, |\bar{S}| \geq n - \lceil n/2 \rceil$, we have $C_S \vdash F^i_S$ and $C_{\bar{S}} \vdash F^j_{\bar{S}}$, for some decision values i and j. Suppose that there exists two schedules σ_0 and σ_1 such that $C_S \overset{\sigma_0}{\vdash} F^0_S$ and $C_{\bar{S}} \overset{\sigma_1}{\vdash} F^1_{\bar{S}}$ (or vice-versa). Then we can apply the schedule $\sigma = \sigma_0 \cdot \sigma_1$ to configuration C, and this results in a configuration where processes in S decide 0, and processes in $\bar{S}$ decide 1 (or vice-versa). This contradicts the consistency of the protocol, and our claim is proved.

By Lemma 2, there is a bivalent initial configuration I. From our claim, without loss of generality, both I_S and $I_{\bar{S}}$ are 1-valent. Let σ be a schedule such that $I \overset{\sigma}{\vdash} F^0$. We denote by I^t the configuration reached from I after the first t steps in σ. Note that $I^0 = I$ and $I^{|\sigma|} = F^0$. Clearly, both $I^{|\sigma|}_S$ and $I^{|\sigma|}_{\bar{S}}$ are 0-valent. Let t be the smallest index such that both I^t_S and $I^t_{\bar{S}}$ are 0-valent. Note that $t > 0$. From our initial claim, and the minimality of t, both I^{t-1}_S and $I^{t-1}_{\bar{S}}$ must be 1-valent.

Let p be the process that performs the atomic step s such that $I^{t-1} \overset{s}{\vdash} I^t$. Suppose p belongs to S, and therefore $I^{t-1}_{\bar{S}} \vdash I^t_{\bar{S}}$. Since $I^t_{\bar{S}}$ is 0-valent, we have $I^t_{\bar{S}} \vdash F^0_{\bar{S}}$. Then we must have $I^{t-1}_{\bar{S}} \vdash F^0_{\bar{S}}$. But $I^{t-1}_{\bar{S}}$ is 1-valent, and this is a contradiction. We obtain a similar contradiction if we assume that p belongs to $\bar{S}$. □

Note that the proof of the theorem holds for any type of protocol, even a probabilistic or a nondeterministic one.

2.3 FAIR SCHEDULERS.

Protocols for asynchronous systems can be viewed as consisting of *rounds.* While in round t, a process sends messages to every other process, and waits until it receives $n - k$ messages sent by distinct processes in round t. The process then changes its state, and starts round $t + 1$. The new state is a function of the old state and the messages received in round t. Processes cannot wait for more than $n - k$ messages since there is always the possibility that all k faulty processes do not send any messages in round t.

Define $R(q, p, t)$ to be the event that p receives a message from q in round t. The progress of the system depends on the joint probability distribution of the $R(q, p, t)$ events, which is determined by the scheduler.

A scheduler is *fair* if the following conditions hold:

(1) For any processes, p and q, and round t, there is a positive constant ϵ such that $\Pr[R(q, p, t)] > \epsilon$.
(2) For any distinct processes r, p, and q, and round t, the events $R(q, r, t)$ and $R(q, p, t)$ are independent.

In particular, these conditions guarantee that, for any round k, there is a constant probability ρ that all processes receive $n - k$ messages from the same set of correct processes.

2.4 AN $\lfloor (n-1)/2 \rfloor$-RESILIENT CONSENSUS PROTOCOL. In this section we describe a k-resilient consensus protocol for a system with a fair scheduler, and $k = 1, 2, \ldots, \lfloor (n-1)/2 \rfloor$. The protocol consists of rounds as in Section 2.3. Both the state of a process and the messages exchanged consist of a *phase number*, a binary *value*, and a *cardinality*. In each phase, a process first sends a message with its state to all the processes, then it waits for messages. When a process receives $n - k$ messages, it considers the sets of messages with value 0 and value 1, respectively. A message with value i and cardinality greater than $n/2$ will be called a *witness for i*. If a process receives a witness for i it changes its value to i (we prove later that no process can receive a witness for both values). Otherwise, it changes its value to the value i with the largest message set. In both cases, it also changes its cardinality to the size of the message set with value i, and then it starts a new phase. A process decides i if it receives more than k witnesses for value i. This indicates that there are enough witnesses for that value in the message system to force the rest of the processes to reach the same decision.

THEOREM 2. *For any k, $0 \leq k \leq \lfloor (n-1)/2 \rfloor$, the protocol described in Figure 1 is a k-resilient consensus protocol for the fail-stop case.*

PROOF. We need the following few definitions. Each execution of the protocol outer loop is called a *phase*. A process is in phase t if at the beginning of this phase its variable *phaseno* has the value t. A message (witness for i) whose *phaseno* field is equal to t is called a t-message (t-witness for i). A process *p decides in phase t* if it sets the decision variable d_p while its *phaseno* variable is equal to t. The value of the variable *var* of process p, when p is at the beginning of phase t, is denoted by var_p^t.

We prove the theorem by showing the protocol's consistency, deadlock-freedom, convergence, and bivalence, in the presence of up to k faulty processors.

Consistency. Let t be the smallest phase in which a process decides. We claim that, for any processes p and q, we cannot have both $witness\_count(0)_p^t > 0$ and $witness\_count(1)_q^t > 0$. Suppose for some i, $witness\_count(i)_p^t > 0$. Then process p, in phase $t - 1$, must have received from some process r a $(t-1)$-witness for i. So r must have received, in phase $t - 2$, more than $n/2$ $(t-2)$-messages with value i. Therefore, if both $witness\_count(0)_p^t > 0$ and $witness\_count(1)_q^t > 0$, since there are at most n processors, there must be a least one processor that sent $(t-2)$-messages with both values. This is impossible in the protocol described in Figure 1, and the claim is proved. From this claim and the description of the protocol, it is now easy to check that a process can never have both $witness\_count(0)$ and $witness\_count(1)$ greater than 0 in the same phase.

Let t be the smallest phase in which a process decides, let us say process p decides 0 in phase t. We prove that no other process q can decide 1.

Since p decides 0 in phase t, we have $witness\_count(0)_p^t > k$. From our claim, we cannot have $witness\_count(1)_q^t > k$. Therefore, if q decides in phase t, it also decides 0.

We now show that all the t-messages sent are of the form $(t, 0, cardinality)$. Since $witness\_count(0)_p^t > k$, process p receives more than $k(t-1)$-witnesses for 0. Consider a process r that sends a t message. Process r must have received $n - k$ $(t-1)$-messages, and one of them must be a $(t-1)$-witness for 0. Then process r

```
process p: k-consensus
        value: integer init($i_p$)
        cardinality: integer init(1)
        phaseno: integer init(0)
        witness_count: array[0..1] of integer init(0)
        message_count: array[0..1] of integer init(0)
        msg: record of
                phaseno: integer
                value: integer
                cardinality: integer
        while (witness_count(0) $\le k$ and witness_count(1) $\le k$)
          message_count := witness_count := 0
          for all q, $1 \le q \le n$, send(q, (phaseno, value, cardinality))
          while (message_count(0) + message_count(1) $< n - k$)
            receive(msg)
            case
              (msg.phaseno = phaseno):
                begin
                  message_count(msg.value) := message_count(msg.value) + 1
                  if msg.cardinality $> n/2$
                  then witness_count(msg.value) := witness_count(msg.value) + 1
                end
              (msg.phaseno > phaseno):
                send(p, msg)
            end
          if there is i such that witness_count(i) > 0
        then value := i
        else if message_count(1) > message_count(0)
            then value := 1
            else value := 0
          cardinality := message_count(value)
          phaseno := phaseno + 1
        end
        let i be such that witness_count(i) > k
        $d_p$ := i
        for all q, $1 \le q \le n$,
          begin
            send(q, (phaseno, value, $n - k$))
            send(q, (phaseno + 1, value, $n - k$))
          end
```

FIG. 1. A k-resilient consensus protocol for the fail-stop case.

increments *witness_count*(0) in phase $t - 1$. From our initial claim, process r sets its value to 0 in phase $t - 1$, and it sends (t, 0, *cardinality*) messages in phase t.

Consider a process q that decides in phase $t + 1$. From the above remark, all the t-messages received by q have value 0, and therefore q must decide 0.

We now prove that all the (t + 1)-messages sent are of the form (t + 1, 0, $n - k$). Consider a process r that sends (t + 1)-messages. From the description of the protocol in Figure 1, we see that if r decides in phase t, the (t + 1)-messages it sends are of the form (t + 1, 0, $n - k$). If r does not decide in phase t, it must have received $n - k$ t-messages in phase t. We already proved that all the t-messages have value 0. So, in phase t, process r sets its value to 0 and its cardinality to $n - k$. Therefore, it sends (t + 1, 0, $n - k$) messages in phase t + 1.

A process r that reaches phase $t + 2$ must have received $n - k$ (t + 1)-messages. From our remark above, all the (t + 1)-messages are witnesses for 0, and therefore r decides 0 in phase $t + 2$.

Since any process that reaches phase $t + 2$ decides 0, no process can ever be in a phase higher than $t + 2$, and no process can decide 1.

Deadlock-freedom. Since processes wait for each other's messages, the protocol might be exposed to deadlocks. We prove that the protocol is deadlock free.

Suppose, for contradiction, the protocol runs into a deadlock. Let D be the set of deadlocked processes. Each process q in D is deadlocked in phase t_q. Let $t_0 = \min_{q \in D} t_q$, and $p \in D$ be a process that is deadlocked in phase t_0. Let S be a set of $n - k$ correct processes. There are two possible cases.

(1) No process in S decides in a phase t, $t \leq t_0 - 2$. By the minimality of t_0, every process in S either decides in phase $t_0 - 1$ or t_0, or it reaches phase t_0 without deciding, in either case it sends t_0-messages to all the processes. Therefore, there will be at least $n - k$ t_0-messages in p's buffer, and p cannot be deadlocked in phase t_0; this is a contradiction.

(2) Some process decides in phase t, $t \leq t_0 - 2$. Let t be the smallest phase in which a process decides. In the proof of the protocol consistency, we showed that no process can ever be in a phase greater than $t + 2$. We also proved that every process that reaches phase $t + 2$ decides. Note that p is deadlocked in phase $t_0 \geq t + 2$. This is a contradiction, and the proof of deadlock-freedom is complete.

Convergence. Let S be a set of $n - k$ correct processes. Suppose no process in S decides in a phase t, $t < t_0$. We prove that there is a fixed θ such that, with probability greater than θ, all the processes in S decide in phase $t_0 + 2$.

Since there are no deadlocks, every process in S will reach phase t_0. Note that, for $t = t_0$, $t_0 + 1$, and $t_0 + 2$, from our assumption of a fair scheduler, there is a positive constant ρ such that, with probability greater than ρ, every process in S receives in phase t the set of $n - k$ t-messages sent by all the processes in S in phase t. In other words, with probability greater than $\theta = \rho^3$, for three consecutive phases all the processes in S exchange messages exclusively among themselves, oblivious to the rest of the system. It is clear from the protocol that, if this happens, then all the processes in S decide in phase $t_0 + 2$.

Bivalence. If all the processes start with the same input value, all the correct processes decide that value within two steps. □

Note that the protocol computes an "approximation" of the majority of the initial input values. If more than $(n + k)/2$ processes start with the same input value, every correct process decides that value in just three phases. If no input value appears in more than $(n + k)/2$ processes, then the consensus value reached is not known a priori.

3. *The Malicious Case*

3.1 THE MODEL. In this section, we investigate a stronger failure behavior of the processes. A *malicious* process can send false and contradictory messages (even according to some malicious design), can fail to send messages, and can change its internal state to any other state.

However, the message system must provide a way for correct processes to verify the identity of the sender of each message. Otherwise, one malicious process can impersonate the whole system, leading the correct processes to conflicting decisions.

The rest of the model is as described in Section 2.1 with the following additional definitions. A schedule is *legal* if all its steps are according to the protocol. A configuration C is *legally reachable* if it is reachable by a legal schedule. Henceforth, we reserve the notation $\vdash$ to denote only transitions by legal schedules.

3.2 A Lower Bound on the Number of Correct Processes

LEMMA 3. *With a k-resilient consensus protocol, for any reachable configuration C, and for any subset S of processes that contains at least n − k correct processes, either* $C_S \vdash F_S^0$ *or* $C_S \vdash F_S^1$ *by some legal schedule.*

PROOF. The malicious processes can behave just like fail-stop processes and die. The proof follows from this observation and the proof of Lemma 1. □

THEOREM 3. *There is no* $\lceil n/3 \rceil$*-resilient consensus protocol for the malicious case.*

PROOF. Suppose there is a $\lceil n/3 \rceil$-resilient protocol. Let S and T be subsets of processes of size $\lfloor 2n/3 \rfloor$ such that $|T \cup S| = n$. Note that $|T \cap S| \leq n/3$. Let C be a legally reachable configuration. All the malicious processes have followed the protocol so far. If they continue to follow the protocol, then there is no way in which they differ from correct processes. Therefore, by Lemma 3, $C_S \vdash F_S^i$ and $C_T \vdash F_T^j$, for some decision values i and j.

We claim that C_S and C_T are either both 0-valent, or both 1-valent. Suppose not, then, without loss of generality, there are legal schedules σ_0 and σ_1 such that $C_S \overset{\sigma_0}{\vdash} F_S^0$ and $C_T \overset{\sigma_1}{\vdash} F_T^1$. Suppose that all the processes in $T \cap S$ are malicious. The following schedule is possible. From C, by schedule σ_0, we first reach a configuration where all the correct processes in S decide 0. Then, the malicious processes in $S \cap T$ change their state and their buffers' contents back to what they were in C, resulting in some configuration C'. The only difference between C'_T and C_T is that in C'_T the buffers of the processes in T may have additional messages (that were added during the execution of σ_0). Since $C_T \overset{\sigma_1}{\vdash} F_T^1$, the processes in T can now follow the legal schedule σ_1 from configuration C', until all the correct processes in T decide 1. This schedule violates the consistency of the protocol, and our claim is proven.

The rest of the proof follows closely the last part of the proof of Theorem 1. Let I be the bivalent initial configuration guaranteed by Lemma 2. From our claim, without loss of generality, both I_S and I_T are 1-valent. Let σ be a legal schedule such that $I \overset{\sigma}{\vdash} F^0$. We denote by I^t the configuration reached from I after the first t steps in σ. Clearly, both $I_S^{|\sigma|}$ and $I_T^{|\sigma|}$ are 0-valent. Let t be the smallest index such that both I_S^t and I_T^t are 0-valent. Note that $t > 0$. From our initial claim, and the minimality of t, both I_S^{t-1} and I_T^{t-1} must be 1-valent.

Let p be the process that performs the atomic step s such that $I^{t-1} \overset{s}{\vdash} I^t$. Assume that p belongs to S. We have $I_S^t \vdash F_S^0$, and therefore $I_S^{t-1} \vdash F_S^0$. However, I_S^{t-1} is 1-valent, and this is a contradiction. We obtain a similar contradiction if we assume that p belongs to T. □

3.3 A $\lfloor (n-1)/3 \rfloor$-RESILIENT CONSENSUS PROTOCOL. In this section, we present a k-resilient consensus protocol for a system with a fair scheduler and $k = 1, 2, \ldots, \lfloor (n-1)/3 \rfloor$ malicious processes. The state of a process consists of a *phase number*, and a binary *value*. As in Section 2.3, the protocol consists of phases in which processes send each other their states. In order to overcome misleading messages from the malicious processes, the state information is sent in the following manner. There are two types of messages: *initial* and *echo*. A process sends to all the processes an initial message with its name and its state. Upon receiving an initial message, every process echoes it back to all the processes. Process p, at phase t, *accepts* a message with value i from process q if it receives more than $(n + k)/2$ messages of the form $(echo, q, i, t)$.

In each phase, a process first sends (through the procedure described above) its state to all the processes, and waits until it accepts messages from $n - k$ processes. Then, it changes its value to the majority of the values of the accepted messages. A process decides i if it accepts more than $(n + k)/2$ messages with value i. We prove that, once a process decides i, thereafter all the other correct processes will have value i.

In the protocol described in Figure 2, processes do not exit the protocol after they decide. This was done for notational convenience only, and can be avoided in the following manner: When process p decides i, it sends to all the processes the message $(initial, p, i, *)$ and echoes of the form $(echo, q, i, *)$ for all q's. These last messages are special, and whenever a process receives them, it sends them back to itself. Once a correct process has decided i, all the correct processes will have value i. Therefore, this procedure will have the same effect as the actual participation of p in the protocol.

THEOREM 4. *For any k, $0 \le k \le \lfloor (n-1)/3 \rfloor$, the protocol described in Figure 2 is a k-resilient consensus protocol for the malicious case.*

PROOF. We show the protocol's deadlock-freedom, consistency, convergence, and bivalence, in the presence of up to k faulty processes. We use the same notation and definitions as in the proof of Theorem 2.

Deadlock-freedom. We have to prove that it is always possible for a process to accept $n - k$ messages. Consider a correct process p in phase t, where t is the smallest phase among correct processes in the system. At least $n - k$ correct processes are in phase t or in a higher phase. Let q be such a process. Process q has already sent a $(initial, q, v, t)$ message to all the other processes. Since there are at least $n - k$ correct processes, p's buffer will receive at least $n - k$ $(echo, q, v, t)$ messages. Since $n - k > (n + k)/2$, then p, at phase t, eventually accepts this message with value v from q. Therefore, p accepts $n - k$ messages from correct processes, and p proceeds to the next phase.

Consistency. Consider any two processes p and q, at some phase t. We claim that, if p and q accept a message from some process r, then these messages must have the same value. Suppose not; then at phase t, p accepts a message with value 0 from r and q accepts a message with value 1 from r. Then more than $(n + k)/2$ processes sent $(echo, r, 0, t)$ messages to p, and more than $(n + k)/2$ processes echoed $(echo, r, 1, t)$ messages to q. Therefore, more than k processes have sent both $(echo, r, 0, t)$ and $(echo, r, 1, t)$. Since there are at most k malicious processes, then at least one correct process has sent both $(echo, r, 0, t)$ and $(echo, r, 1, t)$. From the description of the protocol, correct processes cannot do that, and, hence, a contradiction.

Let t be the smallest phase in which a correct process decides. Let us say process p decides 0 in phase t. Process p must have accepted messages with value 0 from a set S of more than $(n + k)/2$ processes. By deadlock-freedom, any other correct process q will accept, at phase t, messages from $n - k$ processes. Therefore, it must accept messages from more than $(n + k)/2 - k = (n - k)/2$ processes in S. By our claim, the value of the messages accepted by q from processes in S must be 0. So q accepts more than $(n - k)/2$ messages with value 0, and it changes its value to 0.

At phase $t + 1$, all the correct processes will have value 0. Note that it takes at least $(n - k)/2$ messages with value 1 to change the value of a correct process to 1.

```
process p: $k$-consensus
        value: integer init($i_p$)
        phaseno: integer init(0)
        message_count: array[0..1] of integer init(0)
        echo_count: array[1..n:0..1] of integer init(0)
        msg: record of
                type: (initial, echo)
                from: integer
                value: integer
                phaseno: integer
        while(true)
          message_count := 0
          echo_count := 0
        for all $q$, $1 \le q \le n$, send($q$, (initial, $p$, value, phaseno))
        while(message_count(0) + message_count(1) $< n - k$)
          receive(msg)
          if it is the first message received from the sender
            with these values of msg.type, msg.from and msg.phaseno then
              case
                (msg.type = initial):
                  for all $q$, $1 \le q \le n$, send($q$, (echo, msg.from, msg.value, msg.phasno))
                (msg.type = echo and msg.phasno = phasno):
                  begin
                    echo_count(msg.from, msg.value) := echo_count(msg.from, msg.value) + 1
                    if echo_count(msg.from, msg.value) = $(n + k)/2 + 1$
                    then message_count(msg.value) := message_count(msg.value) + 1
                  end
                (msg.type = echo and msg.phaseno > phaseno):
                  send($p$, msg)
              end
          end
          if message_count(1) > message_count(0)
          then value := 1
          else value := 0
          if there is i such that message_count($i$) $> (n + k)/2$
            then $d_p := i$
          phaseno := phaseno + 1
        end
```

FIG. 2. A k-resilient consensus protocol for the malicious case.

Since there are only $k < n/3$ malicious processes, and $k < (n - k)/2$, this can never happen. Therefore, from phase t on, all the correct processes will have value 0 and they can not decide 1.

Convergence. Let S be a set of correct processes that have not decided yet. Suppose no process in S decides in a phase t, $t < t_0$. We prove that there is a fixed θ such that, with probability greater than θ, all the processes in S decide in phase $t_0 + 1$.

Since there are no deadlocks, every process in S reaches phase t_0. From our assumptions on the system behavior, there is θ such that in phases t_0 and $t_0 + 1$ the following happens with probability greater than θ. At phase t_0, every process in S accepts messages from the same set of $n - k$ processes. At phase $t_0 + 1$, every process in S accepts messages only from correct processes. It is clear from the protocol that all the processes in S decide in phase $t_0 + 2$.

Bivalence. If all the processes start with the same input value, within two phases all the correct processes decide that value. □

Note that if $k < n/5$, once a correct process decides, all the other processes also decide within one phase. As in the fail-stop case, this protocol computes an "approximation" of the majority of the initial input values. If more than $(n + k)/2$ correct processes start with the same input value, every process decides that value in just two phases.

4. *Performance Analysis*

In this section, we bound the expected number of phases required to reach agreement in the protocol of Figure 2. Since the protocol of Figure 2 is an $\lfloor (n - 1)/3 \rfloor$-resilient consensus protocol for malicious processes, it is also an $\lfloor (n - 1)/3 \rfloor$-resilient consensus protocol for fail-stop processes. We analyze its performance with both types of processes. The expected running time of failure-free executions is analyzed in Section 4.1, and of executions with failures in Section 4.2.

4.1 FAILURE-FREE EXECUTIONS. In this section, we analyze the expected number of rounds to reach agreement if no failure occurs. We can describe the execution of the protocol as a Markov chain P with states $0, \ldots, n$. The system is at state i if i processes have value 1. For $k = n/3$, we get the following transition probabilities:

$$P_{i,j} = \binom{n}{j} w_i^j (1 - w_i)^{n-j}, \qquad (1)$$

where

$$w_i = \sum_{2n/3 \geq r > n/3} \frac{\binom{i}{r}\binom{n-i}{2n/3 - r}}{\binom{n}{2n/3}}.$$

The absorbing states are $0, \ldots, n/3 - 1$ and $2n/3 + 1, \ldots, n$.

Let E_j be the expected absorption time from state j. It is clear that $E_{n/2} \geq E_{n/2+1} \geq \cdots \geq E_{2n/3+1} = 0$ and $E_{n/2} \geq E_{n/2-1} \geq \cdots \geq E_{n/3-1} = 0$. This gives us a simple intuitive notion of the "distance" of a state from the absorbing states. We can use it to modify our matrix in a way that will increase the expected absorption time. The closer the state is to the center of the matrix, the further it is away from the absorbing states. Therefore, if for $i < j < n/2$ and $l < n/2$, we decrease $P_{n/2 \pm l, n/2 \pm j}$ and increase $P_{n/2 \pm l, n/2 \pm i}$, the resulting matrix will describe a Markov chain with slower convergence rate. As a particular instance of this procedure, we can "identify" one state as another and substitute state $n/2 \pm j$'s row with state $n/2 \pm i$'s row.

We partition the states of the following five groups:

$$A = \left[0, \ldots, \frac{N}{3} - 1\right],$$

$$B = \left[\frac{n}{3}, \ldots, \frac{n}{2} - \frac{l\sqrt{n}}{2} - 1\right],$$

$$C = \left[\frac{n}{2} - \frac{l\sqrt{n}}{2}, \ldots, \frac{n}{2} + \frac{l\sqrt{n}}{2}\right],$$

$$D = \left[\frac{n}{2} + \frac{l\sqrt{n}}{2} + 1, \ldots, \frac{2n}{3}\right],$$

$$E = \left[\frac{2n}{3} + 1, \ldots, n\right].$$

We identify all B's state with state $(n/2) - (l\sqrt{n}/2) - 1$, all C's states with state $n/2$, and all D's with state $(n/2) + (l\sqrt{n}/2) + 1$. Having done that, we can collapse each group to a single state, by adding all the columns in a group to a single column and eliminating the multiple rows. This leaves us with a 5×5 matrix M.

In order to compute M's entries, we use the following approximation: Let the random variable $\mathbf{B}_{(n,p)}$ be the sum of n Bernoulli trials with success probability p, and $j \geq np$. We approximate $\mathbf{B}$'s distribution by the normal distribution, that is,

$$\Pr\{\mathbf{B}_{(n,p)} \geq j\} \approx \Phi\left(\frac{j - np}{\sqrt{np(1-p)}}\right),$$

where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2}\, dt. \tag{2}$$

In state $n/2$ processes can decide 0 or 1 with equal probability, that is, the expected number of the processes that decide 1(0) in the next phase is $n/2$.

$$M_{C,C} = \sum_{j \in C} P_{n/2,j} \approx 1 - 2\Phi(l).$$

By setting $M_{C,A} = 0$ we can just decrease the expected absorption time, which leaves us with $M_{C,B} = \Phi(l)$.

In order to compute row B's entries, we need a bound on $w_{n/2-l\sqrt{n}/2-1}$, the probability of choosing 1 in state $n/2 - l\sqrt{n}/2 - 1$. If we increase that probability, the resulting matrix will have a higher absorption time.

Given a population of n items, b of them are special, and a random sample of size r. Let the random variable $\mathbf{HG}_{(n,b,r)}$ be the number of special items in the sample. $\mathbf{HG}_{(n,b,r)}$ has the hypergeometric distribution. By (1),

$$w_{n/2-l\sqrt{n}/2-1} = \Pr\left[\mathbf{HG}_{(n,n/2-l\sqrt{n}/2-1,2n/3)} > \frac{n}{3}\right]. \tag{3}$$

The hypergeometric distribution is more dense around its mean than the binomial distribution. Therefore,

$$w_{n/2-l\sqrt{n}/2-1} < \Pr\left[\mathbf{B}_{(2n/3,(n/2-l\sqrt{n}/2-1)/n} > \frac{n}{3}\right] < \Phi\left(\sqrt{\frac{2}{3}} \cdot l\right).$$

If we pick l such that $\Phi(\sqrt{2/3} \cdot l) = 1/3$,

$$w_{n/2-l\sqrt{n}/2-1} < \frac{1}{3}. \tag{4}$$

In order to make a transition from B to C, more than $n/2 - l\sqrt{n}/2 - 1$ processes have to change their value to 1, therefore

$$M_{B,C} = \sum_{j \in C} P_{n/2-l\sqrt{n}/2-1,j}. \tag{5}$$

Using (2), we get

$$M_{B,C} \leq \Phi\left(\frac{n/2 - 1\sqrt{n}/2 - 1 - n/3}{\sqrt{2n/9}}\right) \approx \Phi\left(\frac{\sqrt{n} + 3l}{\sqrt{8}}\right), \tag{6}$$

and, using (2) again,

$$M_{B,A} = \sum_{0 \geq j > n/3} P_{n/2 - l\sqrt{n}/2 - 1, j} > \Phi(0) = \frac{1}{2}. \tag{7}$$

Because of the symmetry of (1), and of our partition we know that $M_{B,A} = M_{D,E}$ and $M_{B,C} = M_{D,C}$. we are interested in the absorption time to any absorbing state, and do not care to which one. Therefore, we can collapse state A and E. In the resulting matrix, B and D have exactly the same transition probabilities, and we can collapse them together too. Again, we can decrease probabilities of transition to AE and increase probabilities of transition to C. Thus, yielding the following matrix R:

$$\begin{array}{c} C \\ BD \\ AE \end{array} \begin{bmatrix} 1 - 2\Phi(l) & 2\Phi(l) & 0 \\ \Phi\left(\dfrac{\sqrt{n} + 3l}{\sqrt{8}}\right) & \dfrac{1}{2} - \Phi\left(\dfrac{\sqrt{n} + 3l}{\sqrt{8}}\right) & \dfrac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

Let us denote the leading 2 × 2 submatrix of R by Q. Let $N = (I - Q)^{-1}$. By [4], the expected absorption time from a state in R is given by the sum of the corresponding row in N. Thus

$$I - Q = \begin{bmatrix} 2\Phi(l) & -2\Phi(l) \\ -\Phi\left(\dfrac{\sqrt{n} + 3l}{\sqrt{8}}\right) & \dfrac{1}{2} + \Phi\left(\dfrac{\sqrt{n} + 3l}{\sqrt{8}}\right) \end{bmatrix} \tag{9}$$

and

$$N = \frac{1}{\Phi(l)} \begin{bmatrix} \dfrac{1}{2} + \Phi\left(\dfrac{\sqrt{n} + 3l}{\sqrt{8}}\right) & 2\Phi(l) \\ \Phi\left(\dfrac{\sqrt{n} + 3l}{\sqrt{8}}\right) & 2\Phi(l) \end{bmatrix},$$

and the sum of N's first row is

$$\frac{2\Phi(l) + 1/2 + \Phi((\sqrt{n} + 3l)/\sqrt{8})}{\Phi(l)}. \tag{10}$$

After substituting the value of l, the expected number of phases is 3.6.

4.2 Executions with Failures. We analyze the expected number of rounds to reach agreement in the protocol of Figure 2. As in the previous section, we model the execution of the protocol by a Markov chain M, with states $0, \ldots, n - k$. The system is in state i if i correct processes have value 1. The absorbing states are $0, \ldots, ((n - 3k)/2) - 1$ and $((n + k)/2) + 1, \ldots, n - k$. However, we modify the Markov chain model, since the failures are not governed by the probabilities of P.

We first consider the case of malicious processes. After any state transition of the system, the malicious processes can set their value arbitrarily. The worst that the malicious processes can do is to shift the system to the "further" state from the absorbing states by balancing the number of 1- and 0-messages. This gives us the following transition probabilities:

$$M_{(n-k)/2\pm i,j} = \begin{cases} P_{n/2\pm(i-k),j} & i \geq k, \\ P_{n/2,j} & i < k, \end{cases} \tag{11}$$

where $P_{i,j}$ is as in Section 4.1.

Starting from the balanced state $(n-k)/2$, the probability of making a transition to an absorbing state is given by

$$\sum_{\substack{0\leq j<(n-3k)/2 \\ (n+k)/2<j\leq n-k}} M_{(n-k)/2,j} \approx 2\Phi(l). \tag{12}$$

Therefore, the expected number of transitions to an absorbing state is bounded by $1/(2\Phi(l))$. Therefore, for $k = O(\sqrt{n})$, the expected absorption time is constant. However, for $k = O(n)$, the expected absorption time is $O(2^n)$.

Fail-stop processes cannot shift the state of the system as the malicious processes do. However, after each transition, if the system is biased toward some value v, fail-stop processes that have v as a value can die, thus creating a balanced system with less processes. The number of processes needed to reset a system from a biased state (B's and D's states) to a balanced one is $O(\sqrt{n})$. Since there are only $O(n)$ fail-stop processes and they can die only once, only $O(\sqrt{n})$ transitions can be balanced in this manner. From then on, the system behaves as in the failure-free case. Therefore, the expected absorption time is $O(\sqrt{n})$.

5. *Asynchronous Byzantine Agreement*

A major problem in distributed systems is ensuring reliable broadcasts, commonly known as *Byzantine Agreement* [6], *Unanimity* [2], or *Interactive Consistency* [7]. All previous treatments of Byzantine Agreement deal with a synchronous system of n processes, where up to k processes can be malicious. Some specially designated process is a *transmitter* that sends a *value* to all the rest of the processes. A Byzantine Agreement is achieved if the following holds:

(1) All correct processes agree on the same value.
(2) If the transmitter is correct, all the correct processes agree on its value.

Implicit is also the requirement that the whole system can be viewed to be in one of the following states: "before broadcast," "executing the agreement protocol," and "after broadcast." Thus, queries about the transmitted value can be handled in a consistent manner by any correct process.

When considering asynchronous systems, we can no longer hold such a view of the system. Some correct processes can proceed with the protocol and reach agreement while others may not yet be aware that the protocol has begun. Even if a process receives a message from the transmitter or from other processes, it may be insufficient to start up the process on the protocol. Some threshold of activity is necessary to start up a process, a threshold that guarantees that all the other correct processes will also start the protocol and will agree on the same value.

The following two scenarios illustrate the necessity of such a scheme.

(1) The transmitter is malicious. At time t_0 it sends to k processes 0-messages, to a different set of k processes 1-messages, and none to the rest. All these messages are received at time t_1. After that, the transmitter stops participating in the protocol. If we regard this as a sufficient condition to start up a Byzantine Agreement protocol, then the system can proceed and agree, let us say on 1, at time t_2.

(2) The transmitter is correct and sends 0-messages to all the processes. At time t_1, the same k correct processes as in scenario 1 receive these 0-messages. Also, k malicious processes receive 0-messages, but they treat them as if they were 1-messages. Any other messages from the transmitter will be received only at a time later than t_2. Consider the system during the interval $[t_1, t_2]$. The processes' view of the system is the same as in scenario 1, and therefore they can simulate it and agree on 1 at time t_2, thus violating requirement 2 of the Byzantine Agreement.

There are two ways to overcome this phenomenon. We can restrict the behavior of a malicious transmitter (it will be enough to force it to send $2k + 1$ messages with the same value). Another way, the one we adopt, is to regard certain views of the system as insufficient to start the protocol. Processes may not start, unless presented with a view that guarantees starting up and agreement of all the correct processes.

We say that an asynchronous Byzantine Agreement is achieved if the following holds:

(1) If the transmitter is correct, all the correct processes decide on its value.
(2) If the transmitter is malicious, then either no correct process will decide or they will all decide on the same value.

5.1 A Lower Bound on the Number of Correct Processes

THEOREM 5. *It is impossible to achieve asynchronous Byzantine Agreement with $k \geq n/3$.*

PROOF. Suppose it is possible; since $k \geq n/3$, we can partition the processes to three disjoint sets, A, B and C, of size k or less. Let the transmitter be in A and consider the following scenarios:

(1) The processes in A and B are correct, and the transmitter sends 0-messages. The processes in C are malicious, and they do not send any messages during the protocol. Since the transmitter is correct, the processes in A and B will agree on 0 within some time t.

(2) Only the transmitter is malicious. It sends 0-messages to processes in A and B, and 1-messages to processes in C. Also, messages from C are delayed for a period longer than t before they are received. The processes in A and B have the same view of the system as in scenario 1, and therefore can agree on 0 at time t.

In a similar fashion we can construct scenario 3 with the following properties:

(3) Only the transmitter is malicious. It sends 1-messages to A and C, and 0-messages to B. Messages from B are delayed for a period longer than t'. At time t' the processes in A and C agree on 1.

Now we can combine scenarios 2 and 3 to yield a contradiction:

(4) The processes in A are malicious, the processes in B and C are correct. The processes in A send messages to processes in B as in scenario 2, and to processes in C as in scenario 3. All messages between processes in B and processes in C are delayed for a period longer than $\max(t, t')$. In this scenario, at time $\max(t, t')$, the processes in B will agree on 0 and the processes in C will agree on 1, a contradiction. □

5.2 An Asynchronous Byzantine Agreement Protocol. There are three types of messages in the protocol: *initial*, *echo*, and *ready*. The protocol starts with

```
msg_count: array of [types: 0..1] of integer
msg: record of type: (initial, echo, ready)
value: integer

while(there is no i such that
        msg_count(initial, i) ≥ 1 or
        msg_count(echo, i) > (n + k)/2 or
        msg_count(ready, i) ≥ k + 1)
  receive(msg)
  if it is the first message received from the sender
  with these values of msg.type, msg.from
  then msg_count(msg.type, msg.value) = msg_count(msg.type, msg.value) + 1
end
for all q, send(echo, i)

while(there is no i such that
        msg_count(echo, i) > (n + k)/2 or
        msg_count(ready, i) ≥ k + 1)
  receive(msg)
  if it is the first message received from the sender
  with these values of msg.type, msg.from
  then msg_count(msg.type, msg.value) = msg_count(msg.type, msg.value) + 1
end
for all q, send(ready, i)

while(there is no i such that
        msg_count(ready, i) ≥ 2k + 1)
  receive(msg)
  if it is the first message received from the sender
  with these values of msg.type, msg.from
  then msg_count(msg.type, msg.value) = msg_count(msg.type, msg.value) + 1
end
decide i
```

FIG. 3. An asynchronous Byzantine Agreement protocol.

the transmitter sending *initial* messages. Then processes report to each other the value they received via (*echo*, v) messages. If more than $(n + k)/2$ (*echo*, v) messages are received by a process, it announces it with (*ready*, v) messages. If a process receives $2k + 1$ ready messages of the same value, it decides that value.

THEOREM 6. *The protocol in Figure* 3 *achieves Asynchronous Byzantine Agreement for* $k = 1, \ldots, \lfloor (n-1)/3 \rfloor$ *malicious processes.*

PROOF. We have to show that if some correct process p decides some value, then all the correct processes also decide the same value, and that if the transmitter is correct then they all decide on the transmitter's value.

First, we claim that no two correct processes p and q can send *ready* messages with different values. Suppose this is possible, then p received more than $(n + k)/2$ (*echo*, 1) messages, or a (*ready*, 1) message from a correct process. Similarly, q received more than $(n + k)/2$ (*echo*, 0) messages, or a (*ready*, 0) message from a correct process. In either case, some two correct processes, s and t, received more than $(n + k)/2$ (*echo*, 0) messages, and more than $(n + k)/2$ (*echo*, 1) messages, respectively. Therefore, some correct process r must have sent both (*echo*, 1) and (*echo*, 0) messages. But this is impossible for a correct process. Since decision requires $2k + 1$ *ready* messages with the same value, it is also clear that no two correct processes can decide different values.

Suppose p decides i, then p received $2k + 1$ (*ready*, i) messages. At least $k + 1$ of them were sent by correct processes. Therefore, every correct process will also receive at least $k + 1$ (*ready*, i) messages, and will send its (*ready*, i) message. Thus, at least $n - k$ processes will send (*ready*, i) messages. Therefore, every correct process will receive at least $2k + 1$ (*ready*, i) messages and will decide i.

It is clear that if the transmitter is correct, then all the correct processes will decide on its value. □

6. *Conclusion*

We investigated probabilistic consensus protocols for asynchronous systems with fair schedulers. For a system with fail-stop processors, we showed that $\lceil (n + 1)/2 \rceil$ correct processes are necessary and sufficient for achieving consensus. For a system with malicious processes, we showed that $\lceil (2n + 1)/3 \rceil$ correct processes are necessary and sufficient for achieving consensus.

Probabilistic consensus protocols are also investigated in [1]. The protocols are similar to those given in this paper, but randomization is incorporated in the protocol itself. They have an exponential expected termination time in the fail-stop case, and, in the malicious case, they can overcome up to $n/5$ malicious processes.

These are other examples of problems [5, 8] that do not have a deterministic algorithm, but have probabilistic algorithms that terminate with probability 1. From our analysis in Section 4, it seems that these probabilistic consensus protocols provide a viable solution.

REFERENCES

1. BEN-OR, M. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing* (Montreal, Ont., Canada, Aug.). ACM, New York, 1983, pp. 27–30.
2. DOLEV, D. Unanimity in an unknown and unreliable environment. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science* (Nashville, Tenn., Oct.). IEEE, New York, 1981, pp. 159–168.
3. FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. Impossibility of distributed consensus with one faulty process. *J. ACM 32*, 2 (Apr. 1985), 374–382.
4. ISAACSON, D. L., AND MADSEN, R. W. *Markov Chains Theory and Practice*. Wiley, New York. 1976, pp. 89–100.
5. ITAI, A. AND RODEH, M. Symmetry breaking in distributive networks. In *Proceedings of the 22nd Annual Symposium on Foundation of Computer Science* (Nashville, Tenn. Oct.), IEEE, New York, 1981, pp. 150–158.
6. LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Byzantine Generals problem. *ACM Trans. Prog. Lang. Syst. 4*, 3, (July 1982), 382–401.
7. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2, (April 1980), 228–234.
8. RABIN, M., AND LEHMANN, D. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8th ACM Symposium on the Principles of Programming Languages* (Williamsburg, Va. Jan. 26–28). ACM, New York, 1981, pp. 133–138.
9. SCHLICHTING, R. D., AND SCHNEIDER, F. B. Fail-stop processes: An approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Syst. 1*, 3 (Aug. 1983), 222–238.

RECEIVED APRIL 1984; REVISED APRIL 1985; ACCEPTED JUNE 1985