Information and Computation **150**, 75–111 (1999)
Article ID inco.1998.2770, available online at http://www.idealibrary.com on IDEAL®

Byzantine-Resistant Total Ordering Algorithms*

Louise E. Moser and P. M. Melliar-Smith

Department of Electrical and Computer Engineering,
University of California, *Santa Barbara*, *California 93106*
E-mail: moser@ece.ucsb.edu

Multicast group communication protocols are used extensively in fault-tolerant distributed systems. For many such protocols, the acknowledgments for individual messages define a causal order on messages. Maintaining the consistency of information, replicated on several processors to protect it against faults, is greatly simplified by a total order on messages. We present algorithms that incrementally convert a causal order on messages into a total order and that tolerate both crash and Byzantine process faults. Varying compromises between latency to message ordering and resilience to faults yield four distinct algorithms. All of these algorithms use a multistage voting strategy to achieve agreement on the total order and exploit the random structure of the causal order to ensure probabilistic termination.

1. INTRODUCTION

Modern fault-tolerant distributed systems are often built on multicast group communication protocols. Typically, such protocols impose a causal order on messages [L]. This causal order is readily derived from the acknowledgments used to ensure reliable delivery of messages. In many fault-tolerant distributed systems, information is replicated on several processors to achieve faster access and to protect against faults. Maintaining the consistency of replicated information is, however, a significant programming problem that can be simplified by imposing a total order, rather than a causal order, on messages. The total order ensures that

* A preliminary version of this paper was presented at the 9th International Workshop on Distributed Algorithms, Le Mont Saint Michel, France, September 1995. This research effort was sponsored by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Matériel Command, USAF, under grant number F30602-95-1-0048. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency and Rome Laboratory or the U.S. Government.

The U.S. Government's right to retain a nonexclusive royalty-free license in and to the copyright covering this paper, for governmental purposes, is acknowledged.

0890-5401/99 $30.00

exactly the same updates are applied to the replicated data in the same order at all sites, ensuring consistency and simplifying the application programming.

The Total algorithms presented here are executed independently and concurrently by each process, and incrementally convert a causal order on messages into a total order, without requiring the transmission of any additional messages. These algorithms operate in the presence of asynchrony and communication faults, which may cause different processes to receive messages in different orders, and they tolerate both crash and Byzantine process faults [C]. A crash fault causes a process to stop processing and generating messages. A Byzantine fault allows arbitrary behavior by the faulty process, including generation of messages intended to mislead the other processes. Byzantine faults are of concern, not only to a small number of highly critical fault-tolerant applications, but also to a much larger number of military and civilian applications that must resist deliberate malicious attacks. The Total algorithms ensure that nonfaulty processes construct identical total orders on messages and that non-Byzantine processes construct consistent total orders, provided that the resilience requirements are satisfied.

In prior work [MMA93] we developed two algorithms for deriving a total order on messages from a causal order. Those algorithms tolerate crash faults but not Byzantine faults. The simpler of the two algorithms tolerates k crash faults in an n-process system, where $n > 3k$, while the somewhat more complex algorithm tolerates k crash faults in an n-process system, where $n > 2k$. In this paper, we present four algorithms that tolerate both crash and Byzantine faults. The four algorithms presented here tolerate k_c crash faults and k_b Byzantine faults in an n-process system, where k_c, k_b, and n satisfy the inequalities shown in Table 1. These algorithms are mixed failure mode algorithms that share resilience between crash and Byzantine faults. In the presence of fewer Byzantine faults, the algorithms can tolerate more crash faults, and vice versa, as long as k_c and k_b are not exceeded.

Of these algorithms, the first and second algorithms employ a slightly simpler voting strategy, while the third and fourth algorithms employ a more complex voting strategy that enables them to tolerate more crash faults. In the second and fourth algorithms, an additional preliminary phase converts Byzantine faults into crash faults, allowing the algorithms to tolerate more Byzantine faults. An analysis of the primary performance characteristic, the latency or delay from multicasting a message until it is placed into the total order, presented in Section 9, shows that none of these algorithms dominates any other. For each algorithm, there exist

TABLE 1

The Inequalities That Must Be Satisfied by Each of the Four Algorithms, Expressed in Terms of the Number n of Processes in the System, the Number k_c of Crash Faults, and the Number k_b of Byzantine Faults.

Algorithm 1	$n > 3k_c + 5k_b$
Algorithm 2	$n > 3k_c + 3k_b$
Algorithm 3	$n > 2k_c + 5k_b$
Algorithm 4	$n > 2k_c + 3k_b$

values of k_c, k_b, and n such that the algorithm achieves a lower latency than any of the other algorithms.

Systems capable of tolerating Byzantine faults must be able to confirm the source of a message and to detect messages that have the same header but different contents and/or acknowledgments within the message. A typical implementation might use a digital signature to substantiate the source of a message. It might also embed a digest of the content of each message within the identifier of the message, so that acknowledgments of messages with different contents will be regarded as acknowledgments of distinct messages. The acknowledgment and retransmission mechanisms can then ensure that all versions of a message are delivered to all destinations. Digital signatures and digests are computationally expensive, but do not require the additional rounds of message exchange of alternative strategies.

An interesting aspect of the Total algorithms is that they are, as far as we are aware, the only available truly fault-tolerant total ordering algorithms. All other multicast total ordering protocols known to us become blocked transiently if even one process fails. A low-level fault detector algorithm [CT, CHT] detects the faulty process, and a membership algorithm reconfigures the system to exclude the faulty process, unblocking the protocol. During the operation of the membership algorithm, no new messages can be ordered and delivered. With high probability, the Total algorithms continue to order messages even though some of the processes are faulty, provided that the resilience requirements are satisfied. Message transmission and acknowledgment scenarios exist for which the Total algorithms are unable to order and deliver messages, but these occur with negligible probability. Correspondingly, for all known membership algorithms that are not based on a total order, a small probability exists that the membership algorithm will deliver no membership or a trivial singleton membership.

In [MMA94] we provided four alternative fault detection and reconfiguration algorithms that operate on top of the Total algorithms, exploiting the total order generated by them. These membership algorithms, adapted to Byzantine faults, allow the values of k_c, k_b, and n to be reduced as processes fail, and to be increased as processes recover or as new processes are introduced, thus operating continuously with the optimal resilience for the available configuration.

2. RELATED WORK

The problem of determining a consistent total order on messages in a distributed system is related to the problems of agreement and consensus in such systems. Research into these problems was initiated by Pease, Shostak, and Lamport [PSL] with an agreement algorithm for a synchronous system in which the behavior of faulty processes can be Byzantine, i.e. completely arbitrary even to the extent of being malicious. For their Byzantine model, they showed that, in a synchronous system, the consensus problem admits a solution if and only if $n > 3k$, where n is the number of processes and k is the number of Byzantine processes. Their strategy requires the same number of rounds and the same number of messages, regardless of whether or not faults actually occur in the system.

For asynchronous systems, the agreement and consenus problems are more difficult than for synchronous systems. Fischer, Lynch, and Paterson [FLP] demonstrated that consensus is impossible to achieve in an asynchronous system, even for crash faults and just one faulty process. Several researchers [B, Ra] have developed explicitly randomized protocols for consensus in an asynchronous system. In those protocols, a randomizer selects one of two or more alternative statements, the choice of which cannot be predicted by the faulty processes. Employing a different approach, Bracha [BT] developed consensus protocols for asynchronous systems in which the randomization is derived implicitly from the communication medium rather than explicitly from a randomizer. Our algorithms are also of this type. All of these randomized algorithms are demonstrated to terminate with probability 1.

Typically, practical systems must make many consensus decisions. The cost of a single consensus decision in isolation is high, but the cost per decision can be substantially reduced for a sequence of decisions. Bar-Noy, Deng, Garay, and Kameda [BDGK] investigated algorithms for such sequences of consensus decisions with Byzantine processes for synchronous systems, while Bar-Noy and Dolev [BD] gave an algorithm for finding a sequence of consensus decisions in a pipeline using single-bit messages, also restricted to synchronous systems. Gopal and Toueg [GT] investigated the sequence agreement problem for both synchronous and asynchronous systems, with Byzantine faults in the synchronous case, but restricted to omission faults in the asynchronous case. The Total algorithms that we have developed involve a sequence of consensus decisions, each determining a message to be placed next in the total order.

The four Total algorithms presented here are multiple failure mode algorithms that can withstand various combinations of crash and Byzantine faults. Multiple failure mode consensus algorithms for synchronous systems have been considered by other researchers [GP, M, TP]. The second and fourth Total algorithms essentially convert Byzantine faults into crash faults and then order messages using an algorithm that is tolerant to crash faults. This technique has also been employed by other researchers [BN, B, S, TPS]. Many practical systems [ADKM, BV, KT, MMA90, MMABL, VBM] use consensus and total ordering algorithms that lack tolerance to crash and Byzantine faults; they combine those algorithms with fault detectors [CT, CHT] that detect crashed processes and membership algorithms that remove them from the system. Such an approach can achieve lower costs when no faults occur, but the cost of consensus or ordering decisions may be quite high in the presence of faults. Some total ordering algorithms order messages in the presence of network partitioning faults [DKM, CHD, MMABL]. The Total algorithms are, however, primary component algorithms that do not admit network partitioning faults.

The Rampart system of Reiter [Re] includes a multicast protocol for total ordering of messages in the presence of Byzantine faults. The protocol uses digital signatures and an echo strategy. Under low loads, the echo strategy is quite expensive, both computationally and in the number of messages that must be multicast. Under high loads, several small messages, even from different sources, can be packed into a single reliable multicast message, which reduces the overhead.

However, the latency of the Rampart total ordering protocol is still significantly higher than that of the Total algorithms, which do not require messages to be transmitted twice and do not have to wait for echoes from every process. Recently, considerable efforts have been made to improve the performance of Byzantine total ordering algorithms [KMM].

3. THE MODEL

We consider an asynchronous distributed system with n processes, where $n \geqslant 2$, in which processes communicate by broadcasting messages. The system is asynchronous in that no bound can be placed on the time required for a computation or for communication of a message. Process and communication faults can occur.

A *faulty* process may either have crashed or be Byzantine. We consider a process to have crashed if it makes no further broadcasts after a certain point in its execution of the algorithm, or if any of its messages is not eventually received by some nonfaulty process. A Byzantine process cannot be relied upon to execute the algorithms correctly and can disrupt the usual causal order on messages. However, we assume that a Byzantine process does not send messages that cause other processes to become corrupted (e.g., viruses).

Each broadcast message has a header consisting of a process identifier and a message sequence number. We let m_{pi} denote the ith message in the sequence of messages broadcast by process p. Acknowledgments for individual messages are piggybacked on the messages when they are broadcast.

The Total algorithms convert a Byzantine causal order on messages into a total order on messages that is identical at all nonfaulty processes in the system. The Byzantine causal order is defined in terms of the follows relation, which is derived from acknowledgments in messages. A message m' from process q follows a message m from process p if m' acknowledges m or m' acknowledges a message m'' that follows m, where $m'' \neq m$ and $m'' \neq m'$. This implies that the follows relation is transitive (Property 2 of the formal definition of Byzantine causal order below). Furthermore, each message follows itself, which implies that the follows relation is reflexive (Property 1). Liveness requires that every nonfaulty process must generate further messages that follow the messages from every nonfaulty process (Property 6).

A message m' from a non-Byzantine process q acknowledges a message m only if q has received all messages that m follows before it broadcast m'. The construction of the follows relation is incremental. Knowledge of the follows relations, between m and the messages m follows, allows determination of the acknowledgment relation between m' and m and, thus, of the follows relation between m' and m. There is no circularity in the definition of the follows relation, because the acknowledgment relation between m' and m does not depend on the follows relation between m' and m.

A non-Byzantine process q is assumed to receive each message m it broadcasts and to acknowledge that message in the next message it broadcasts. Consequently, a message m' from a non-Byzantine process q follows all messages broadcast earlier by q (Property 4). This requirement cannot be assured for messages from Byzantine

processes, because such a process may have transmitted multiple messages that acknowledge the same or different prior messages, and it may even have transmitted messages of which other processes are unaware.

We assume that a digital signature mechanism in the underlying multicast communication protocol prevents a Byzantine process from originating a message purporting its source to be some other process. Furthermore, we assume that a digest mechanism ensures that, if a Byzantine process sends two different messages, purporting that they are the same message, then the digest enables the destinations to recognize the messages as distinct. The digest is included in the message and also in the acknowledgments for the messages, allowing processes to recognize such messages as distinct messages, even if they have not received one or both of those messages.

A Byzantine process can originate a message that occurs within a (nontrivial) cycle in the causal order. A non-Byzantine process cannot originate a message that occurs in a cycle (Property 3), because a message m from a non-Byzantine process cannot acknowledge a message that itself acknowledges m. A nonfaulty process executing one of the Total algorithms does not advance a message to the total order unless it has already advanced to the total order all of the messages that precede that message in the causal order. Thus, a nonfaulty process does not advance to the total order any message in a cycle, or any message that follows a message in a cycle. The nonfaulty processes execute a cycle detection algorithm in their processing of messages and acknowledgments; thus, any message from a non-Byzantine process does not acknowledge a message involved in a cycle (Property 5). If a Byzantine process ever participates in a cycle, then none of its subsequent messages will be ordered and, when the cycle has been detected, the Byzantine process will be removed from the configuration and never subsequently readmitted.

A Byzantine process can originate two or more concurrent messages with the same header, but with different contents and/or acknowledgments, neither of which follows the other and possibly each of which purports to follow different messages. We call such messages *mutants*. Although mutant messages are allowed to occur in the causal order, the voting strategy of the first and third Total algorithms is resilient to them, and the voting strategy of the second and fourth Total algorithms allows only one message from a set of mutants to vote. For all four Total algorithms, only one message is ordered from each set of mutants. When a second mutant message is about to be ordered, the Byzantine process that originated that message is removed from the configuration.

Thus, we let M be the set of messages of a particular execution of the system, and define a *Byzantine causal order* on M in terms of the follows relation, given below.

Byzantine Causal Order. 1. Each message m_1 follows itself.

2. If m_2 follows m_1 and m_3 follows m_2, then m_3 follows m_1.

3. If m_1 and m_2 are distinct messages, m_2 follows m_1, and m_2 is originated by a non-Byzantine process, then m_1 does not follow m_2.

4. If m_1 and m_2 are both originated by the same non-Byzantine process, then either m_1 follows m_2 or m_2 follows m_1.

5. If m_1 and m_2 are distinct messages, m_1 follows m_2, m_2 follows m_1, and m_3 is originated by a non-Byzantine process, then m_3 does not follow m_1 or m_2.

6. If m_1 is originated by a nonfaulty process p, then for every nonfaulty process q there exists a message m_2 from q such that m_2 follows m_1.

When restricted to the messages from non-Byzantine processes, this causal order is the partial order defined by Lamport [L]. Note that the above properties guarantee that messages originated by a given non-Byzantine process are totally ordered.

A *prefix A of a Byzantine causal order* on M is a subset of the messages of M, together with the follows relations between them such that, if m' is an element of A and m' follows m, then m is also an element of A. The *size* of a Byzantine causal order prefix A is the number of messages in A.

A *total order* relation on a set of messages satisfies the reflexive, antisymmetric, transitive, and comparable properties. A *total order prefix* is a finite set of messages that is totally ordered and that can, thus, be represented as a finite sequence $m_{p_1 i_1}, ..., m_{p_t i_t}$.

An *internal state* of a process consists of a prefix of the Byzantine causal order and a prefix of the total order. The *initial state* of a process consists of an empty prefix of the Byzantine causal order and an empty prefix of the total order.

As each nonfaulty process receives messages, it extends its Byzantine causal order prefix. Sometimes receipt of a message may result in the addition of no messages to the causal order prefix, while receipt of another message may lead to the inclusion of several messages.

An *execution step* consists of a process's extending its Byzantine causal order prefix A by adding one message m, together with the follows relations between m and the messages in A, and its total order prefix by zero or more messages. A step is *applicable* to an internal state if all of the messages that m follows are in the Byzantine causal order prefix of that state. Hereafter, we only consider steps that are applicable.

The asynchronous nature and faulty behavior of the processes and the communication medium are reflected in the Byzantine causal order that is input to the algorithm and in the order in which messages are supplied to a process to extend its causal order prefix. A process has no control over which message extends its causal order prefix in a step, and it cannot determine whether any future extension of its causal order prefix will involve a message that follows a particular message. A message from a faulty process may be followed by no other message and, thus, may be supplied to one process an arbitrary number of steps after it has been supplied to another process, if at all.

The Total algorithms are designed around the number k_c of crash processes that can be tolerated and the number k_b of Byzantine processes that can be tolerated. For each of our algorithms, k_c and k_b must satisfy a particular inequality constraint. If the actual number of crash processes is at most k_c and the actual number of Byzantine processes is at most k_b, then the Total algorithms extend the total order with probability 1. If the actual number of crash and Byzantine processes exceeds $k_c + k_b$, but the actual number of Byzantine processes is at most k_b, then

the algorithms may block because they may be unable to obtain the required number of votes. If the actual number of Byzantine processes exceeds k_b, then the algorithms may give incorrect answers in that different total orders may be generated by two nonfaulty processes.

A Byzantine causal order is *admissible* if it satisfies the following Liveness and Fairness properties. These properties consider messages from nonfaulty processes and imply that a faulty process cannot preclude communication between nonfaulty processes. Byzantine causal orders with these properties are readily obtained from the acknowledgments in broadcast messages and an appropriate communication medium.

Liveness. 1. For each message m from a nonfaulty process p, process p eventually obtains a prefix A such that $m \in A$.

Let p and q be nonfaulty processes and let m be a message from p. We say that a message m' from q *directly follows* m if and only if m' follows m and there does not exist a message m'' such that m' follows m'' and m'' follows m, where $m'' \neq m$ and $m'' \neq m'$. Note that the directly follows relation is neither transitive nor reflexive.

For nonfaulty processes p and q, $p \neq q$, we let $\mathrm{E}(m, p, q)$ be the event that there exists a message m' from q that directly follows the message m from p.

Fairness. 1. $0 < \epsilon < \text{probability}(\mathrm{E}(m, p, q)) < 1$

2. $\mathrm{E}(m, p, q)$ is independent of the set of events $\{\mathrm{E}(\bar{m}, r, s) \mid m \neq \bar{m} \text{ or } q \neq s\}$.

These Fairness properties formalize an assumption of no bias in the selection of processes to broadcast or in the choice of processes to receive messages. The intuition underlying the first Fairness property is that, for each nonfaulty process q, the event that message m from a nonfaulty process p is directly followed by a message from q is neither guaranteed nor precluded from happening. Similarly, the second Fairness property requires that such events neither force nor preclude each other. These Fairness properties must be assumed for liveness, but they are not required for safety. If the Fairness properties do not hold, the Total algorithms are sound but may block.

Of course, these assumptions restrict the class of asynchronous behaviors being considered and, thus, the Byzantine causal orders input to the Total algorithms. Robust communication mechanisms, involving point-to-point links, spread-spectrum codes or ATM switches, provide either independent communication paths or enforced fairness, ensuring that a faulty process cannot preclude communication between other processes. Such communication media match the assumptions quite well. The assumptions cannot, however, be substantiated on an unswitched Ethernet, where a single Byzantine process could transmit continuously and prevent all communication between nonfaulty processes.

Observations indicate that these assumptions are reasonable in the absence of Byzantine faults, but each application must be examined individually to determine that its patterns of message generation do indeed satisfy the fairness requirements.

The validity of the Total algorithms depends on showing

Partial Correctness. 1. The total orders determined by any two non-Byzantine processes are consistent; i.e., if any non-Byzantine process determines that m is the ith message of the total order, then no non-Byzantine process determines that m' is the ith message, where $m' \neq m$.

2. The total order is consistent with the Byzantine causal order; i.e. if m' follows m in the total order of any non-Byzantine process, then m does not follow m' in the Byzantine causal order, where $m' \neq m$.

Probabilistic Termination. 1. The probability that a nonfaulty process p places an ith message in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.

2. For each message m broadcast by a nonfaulty process q, the probability that a nonfaulty process p places m in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.

Absolute termination is unfortunately precluded by the impossibility result [FLP].

4. THE TOTAL ORDERING ALGORITHMS

The input to the Total algorithms is a Byzantine causal order of messages, and the output is a total order of messages. The algorithms accept the Byzantine causal order and extend the total order, incrementally. Only information derived from the Byzantine causal order is used to construct the total order; no additional communication between processes is necessary.

From the Byzantine causal order prefix, a nonfaulty process incrementally constructs the total order prefix using one of the Total algorithms. The messages in the causal order prefix that have already been advanced to the total order play no part in the further extension of the total order prefix (other than to identify the second mutant) and are effectively removed from the causal order. Some of the messages in the causal order prefix follow other messages that have not yet been advanced to the total order; such a message cannot be the next message to extend the total order prefix.

A *candidate message* is a message in a process's Byzantine causal order prefix that is not yet in the total order and that only follows messages already in the total order (other than itself). A set of candidate messages is a *candidate set*. Thus, messages that occur in a cycle cannot be candidate messages. However, mutant messages that are not in a cycle can be candidate messages and, even, members of the same candidate set. With the exception of mutants, at most one candidate message from a given process will be considered for the next advancement to the total order.

As a process advances a candidate set from its causal order to the total order, the messages in that set are removed from further consideration and the candidate set for the next extension is automatically determined. Even though two different processes may consider different candidate sets for a given extension of the total

order, the Total algorithms ensure that the two processes decide to extend the total order with the same candidate set.

Messages in the Byzantine causal order prefix (except those in a cycle) provide votes on the candidate sets. These votes are not contained explicitly in the messages, but are deduced from the causal relationships between messages. A nonfaulty process decides to advance a candidate set to the total order based on the votes of the messages in its causal order prefix. Each process makes its own decisions independently of the other processes. Even though messages may be delayed or lost and, thus, different nonfaulty processes may have different Byzantine causal order prefixes, the nonfaulty processes must decide on the same message by which to extend the total order even though other processes may be Byzantine.

An *execution step* consists of adding one message to the Byzantine causal order prefix and executing one of the Total algorithms. In a step, all candidate sets that can be constructed from the candidate messages in the causal order prefix are considered. A step may result in a decision to extend the total order prefix. Alternatively, a step may result in no candidate set's obtaining a favorable decision and, thus, no extension to the total order prefix. In the next step, with an additional message in the causal order prefix, another attempt is made to extend the total order prefix.

In a step, each candidate set is voted on separately and independently. Voting on a candidate set takes place in a sequence of stages, and each candidate set has its own sequence of stages. Voting proceeds sequentially through the stages for a particular candidate set, but concurrently on all of the candidate sets, since it cannot be predetermined which message is able to vote for which candidate set at which stage.

At stage 0, the vote of a message on a candidate set depends on which candidate messages that message follows. In particular, a candidate message votes for the set containing only itself and against all sets that do not contain itself; it does not vote on any other set at stage 0. At stage i, $i > 0$, the vote of a message on a candidate set depends on whether that message follows "enough" (defined below) messages that vote at stage $i-1$. The message itself may be included in the number of messages required to vote since each message follows itself in the causal order.

A message in the causal order may not vote at a stage i, $i \geqslant 0$, if a previous message from its source votes at stage i. At most one message in a set of mutant messages may vote on a particular candidate set at a particular stage; a process chooses the first such message that it received but it could choose any of those mutants. Messages that occur in a cycle may not vote.

A message may be unable to vote on a candidate set at a stage if the Voting Criteria (given below) are not satisfied. Conversely, a message may be able to vote on a candidate set at several stages if the Voting Criteria are satisfied for each of those stages.

The lth decision of a process p in favor of a candidate set for inclusion in the total order determines a set S^l of candidate messages by which p extends its $(l-1)$th total order prefix to obtain its lth total order prefix. The initial candidate set and the initial total order prefix are empty. The messages of the candidate set S^l are included in the total order, following the messages in the $(l-1)$th total order prefix in a deterministic order, such as lexicographic order of process identifiers.

5. THE $3k_c + 5k_b$ TOTAL ALGORITHM

For this algorithm, we assume that the number n of processes, the number k_c of crash processes, and the number k_b of Byzantine processes satisfies the constraint $3k_c + 5k_b < n$. The algorithm is defined by the Eligibility, Voting and Decision criteria given below; these criteria determine whether a set S of candidate messages is chosen for inclusion in the total order.

The number of votes required for a further vote and the number of votes required for a decision are at least N_v and N_d, where

$$N_v = (n - k_c - k_b)/2, \qquad N_d = (n + k_c + 3k_b + 1)/2.$$

Eligibility Criteria. At stage i, where $i \geqslant 0$,

- a non-Byzantine process determines that a message m is eligible to vote on S if
  - — the process has obtained a prefix A of the Byzantine causal order such that $m \in A$, and
  - — no previous message from the source of m has voted on S at stage i.

Voting Criteria. At stage 0,

- a message votes for S if that message follows every message in S and it follows no other candidate message (a candidate message votes for the set containing only itself);
- a message votes against S if that message follows any candidate message other than those in S (a candidate message votes against all sets of which it is not a member).

At stage i, where $i > 0$,

- a message votes for S if
  - — it follows at least two messages that vote on S at stage $i-1$,
  - — it follows at least N_v messages that vote for S at stage $i-1$, and
  - — it follows fewer messages that vote against S than vote for S at stage $i-1$.
- a message votes against S if
  - — it follows at least two messages that vote on S at stage $i-1$,
  - — it follows at least N_v messages that vote against S at stage $i-1$, and
  - — it does not vote for S at stage i.

Decision Criteria. At stage i, where $i \geqslant 0$,

- a non-Byzantine process decides for S if
  - — it determines that at least N_d messages vote for S at stage i, and
  - — for each proper subset of S, it decides against that proper subset.

- a non-Byzantine process decides against S if
  - — it determines that at least N_d messages vote against S at stage i, or
  - — it decides for a proper subset of S.

The values of N_v and N_d and the constraint $3k_c + 5k_b < n$ are derived from the following properties: Votes and decisions do not conflict ($N_v + N_d > n + k_b$), decisions do not conflict ($N_d + N_d > n + k_b$), stages of voting advance ($N_v + N_v - 1 \leqslant n - k_c - k_b$), and decisions are feasible ($N_d \leqslant n - k_c - k_b$).

5.1. Example

Consider a system of $n = 6$ processes executing the $3k_c + 5k_b$ Total algorithm which tolerates a single Byzantine fault, i.e. $k_c = 0$ and $k_b = 1$. In such a system, $N_v = 2.5$ and $N_d = 5$. Consider the causal order prefix shown in Fig. 1. Here the candidate messages are a_1, e_1, and f_1, shown in heavy bold. Note that, at stage 0, the candidate message e_1 does not vote on $\{e_1, f_1\}$ and, similarly, for f_1. The messages b_1, c_1, d_1, e_2, and f_2 all follow both e_1 and f_1 and do not follow the other candidate message a_1. Thus, these five messages all vote for $\{e_1, f_1\}$ at stage 0, and these five votes suffice for a decision in favor of $\{e_1, f_1\}$.

Consider now the scenario shown in Fig. 2 in which process f is Byzantine. Here, process f has sent a mutant message f'_2 to process c. This scenario represents c's view of the Byzantine causal order; the other processes see the Byzantine causal order shown in Fig. 1. Note that the mutant message f'_2 sent to c votes against $\{e_1, f_1\}$ because f'_2 follows the candidate message a_1. Consequently, process c has only four stage 0 votes, too few for a decision but enough for a further stage of

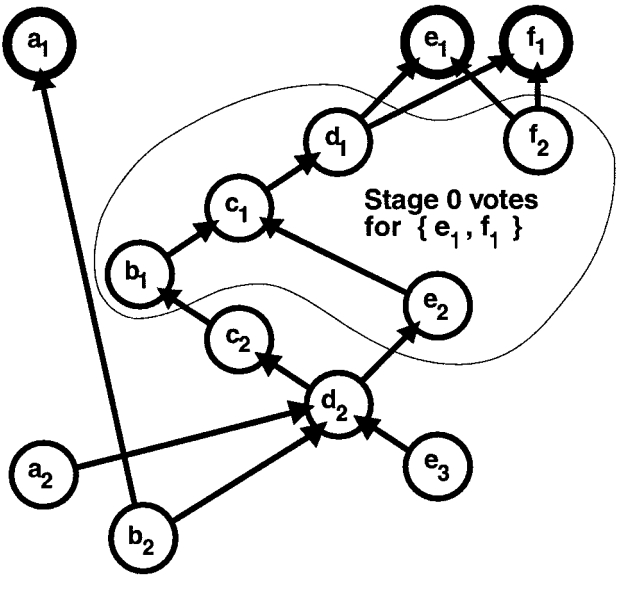

FIG. 1. A scenario for $n = 6$ processes executing the $3k_c + 5k_b$ Total algorithm with $k_c = 0$ and $k_b = 1$. Here $N_v = 2.5$ and $N_d = 5$.

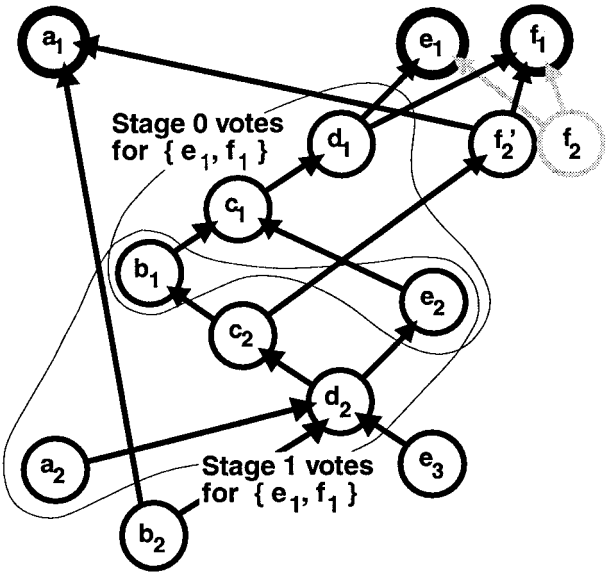

FIG. 2. A variation of the scenario shown in Fig. 1, as seen by process c. Here, a Byzantine process f has generated two mutant messages. Process c receives a mutant f'_2, while all of the other processes receive the same message f_2 as in the previous scenario.

voting. Note that, at stage 1, b_1 and e_2 voted for $\{e_1, f_1\}$ at stage 0, but they can also vote for $\{e_1, f_1\}$ at stage 1. Thus, process c observes stage 1 votes from a_2, b_1, c_2, d_2, and e_2 for $\{e_1, f_1\}$, enough for a decision at stage 1.

The Total algorithms ensure that, if any nonfaulty process can decide for a candidate set at some stage, then any other process must see enough votes for that candidate set at that stage to ensure that it can decide for that candidate set at the next stage.

5.2. Proof of Correctness

5.2.1. Partial Correctness

The proof of partial correctness involves showing that if two non-Byzantine processes p and q make lth extensions to their total order prefixes, then the candidate sets they choose for those extensions are identical and, thus, their lth total order prefixes are identical. The proof is by induction on l. The base case is trivial since, if $l = 0$, then all of these sets are empty. The most interesting part of the proof constitutes establishing the inductive step.

Lemma 5.1 is important because it stipulates that the only information used by the algorithm to determine the vote of a message is the Byzantine causal order. By "determine the vote of a message" we mean determine if the message voted for or against a candidate set or was unable to vote because it did not follow enough messages that voted on the candidate set.

LEMMA 5.1. *Let p and q be non-Byzantine processes. If p and q each determine the vote of a message m on a candidate set S for the lth extension of the total order at stage i, then they determine the same vote of m on S at stage i.*

Proof. The proof is by induction on i. If process p determines the vote of a message m on a candidate set S at stage 0 then, by the Eligibility criteria, p has obtained a Byzantine causal order prefix A that contains message m and all of the messages that m follows and similarly for process q. Since the vote of a message m on a candidate set S at stage 0 is determined by the messages that m follows, p and q determine the same vote of m.

The proof of the inductive step is similar and depends on the Eligibility criteria and the fact that the vote of a message at stage i, where $i > 0$, is determined by the votes of the messages that m follows at stage $i-1$. ▮

LEMMA 5.2. *Each non-Byzantine process p broadcasts at most one message that votes on a candidate set S at stage i; that message does not vote both for and against the candidate set S at stage i.*

Proof. The first statement follows from the Eligibility criteria that a message is eligible to vote on a candidate set S at stage i only if no previous message from its source has voted on S at stage i. The second statement follows from the Voting criteria. ▮

Lemmas 5.1 and 5.2 will be used in subsequent proofs without further reference.

LEMMA 5.3. *If message m follows at least N_d messages that vote for (against) a candidate set S at stage $i-1$, then no message m′ votes against (for) S at stage i, where $i > 0$.*

Proof. If m follows messages from at least $N_d = (n + k_c + 3k_b + 1)/2$ distinct processes that vote for S at stage $i-1$, then messages from at most $n - N_d + k_b = n - (n + k_c + 3k_b + 1)/2 + k_b < (n - k_c - k_b)/2 = N_v$ processes vote against S at stage $i-1$ since the k_b Byzantine processes can broadcast messages that vote both for and against S. If m' votes against S at stage i, then m' follows messages from at least N_v distinct processes that vote against S at stage $i-1$, which is a contradiction. ▮

LEMMA 5.4. *If message m votes against (for) a candidate set S at stage j then, for each i such that $0 \leqslant i \leqslant j$, m follows a message that votes against (for) S at stage i.*

Proof. The proof is by a simple induction on j using the Voting criteria. ▮

LEMMA 5.5. *If no message from a non-Byzantine process votes against (for) a candidate set S at stage i, then no message from any process votes against (for) S at stage j, where $j \geqslant i$. Likewise, if some process p decides for (against) a candidate set S at stage i, then no message votes against (for) S at stage j, where $j > i$.*

Proof. To prove the first statement, we note that if a message m from a non-Byzantine process votes against a candidate set S at stage j then, by Lemma 5.4, m follows a message that votes against S at stage i, which is a contradiction. The

second statement follows from the Decision criteria, Lemma 5.3, and the first statement. ▮

Propositions 5.1–5.5 below constitute a major part of the proof and are used in proving Theorem 5.1, which establishes consistency of decisions among the processes.

PROPOSITION 5.6. *Let p and q be non-Byzantine processes. If p decides for a candidate set S for the lth extension of the total order, then q does not decide for a proper subset S′ of S for the lth extension.*

Proof. The proof is by induction on the cardinality C of S. If $C=1$, the statement holds since, by the Voting and Decision criteria and a simple induction, no process decides on the empty candidate set.

Assume that the statement holds for candidate sets S of cardinality less than C. We argue by contradiction, assuming further that p decides for S and that q decides for a proper subset S' of S at some stage j. Thus, q determines that messages from $N' \geqslant N_d$ distinct processes vote for S' at stage j.

By the inductive assumption, since q decides for a proper subset S' of S, p does not decide for a proper subset S'' of S'. Thus, since p decides for S, p decides against S' at some stage i for the reason that p determines that messages from $N \geqslant N_d$ processes vote against S' at stage i.

If $i=j$, then $N+N' \geqslant N_d+N_d=n+k_c+3k_b+1>n+k_b$ yields a contradiction, since only Byzantine processes can broadcast messages that vote both for and against S' at stage j. If $j>i$ then, by Lemma 5.5, since p decides against S' at stage i, no message votes for S' at stage j, which is a contradiction. Similarly, if $i>j$, we obtain a contradiction by applying Lemma 5.5 to process q with i and j interchanged. ▮

PROPOSITION 5.2. *Let p and q be non-Byzantine processes. If p decides for (against) a candidate set S for the lth extension of the total order, then q does not decide against (for) S for the lth extension.*

Proof. Assume that p decides for S at stage i and that q decides against S at stage j. Since p decides for S at stage i, p determines that messages from $N \geqslant N_d$ distinct processes vote for S at stage i of which at most k_b are Byzantine. Furthermore, by Proposition 5.1, q does not decide for a proper subset S' of S. Thus, q decides against S at stage j for the reason that q determines that messages from $N' \geqslant N_d$ processes vote against S at stage j of which at most k_b are Byzantine.

If $i=j$, then $N+N' \geqslant N_d+N_d=n+k_c+3k_b+1>n+k_b$ yields a contradiction, since only Byzantine processes can broadcast messages that vote both for and against S at stage j. Otherwise, a contradiction is reached by Lemma 5.5. ▮

LEMMA 5.6. *Let S and S′ denote candidate sets for the lth extension of the total order such that there exist $s \in S$, $s \notin S'$ and $s' \in S'$, $s' \notin S$. If message m from process p votes for S at stage i, then m or a previous message from p votes against S′ at stage i.*

Proof. The proof is by induction on i. If message m from process p votes for S at stage 0, then it follows the candidate message $s \in S$. Since m follows s and $s \notin S'$, by the Voting criteria, m or a previous message from p votes against S' at stage 0, which establishes the base case.

We now assume the statement for $i-1$. Let m follow N messages that vote for S at stage $i-1$, $\bar{N}$ messages that vote against S at stage $i-1$, N' messages that vote for S' at stage $i-1$, and $\bar{N}'$ messages that vote against S' at stage $i-1$. Since m votes for S at stage i, $N \geqslant N_v$, $N > \bar{N}$, and $N + \bar{N} \geqslant 2$. Thus, $N \geqslant 2$. By the inductive assumption, if a message votes for S at stage $i-1$, that message or a previous message from its source votes against S' at stage $i-1$. Thus, $\bar{N}' \geqslant N$. Similarly, if a message votes for S' at stage $i-1$, that message or a previous message from its source votes against S at stage $i-1$. Thus, $\bar{N} \geqslant N'$. Consequently, $\bar{N}' \geqslant N_v$, $\bar{N}' > N'$ and $N' + \bar{N}' \geqslant 2$. Thus, m or a previous message from p votes against S' at stage i unless a previous message from p votes for S' at stage i.

Now suppose that there exists such a message m' from p that votes for S' at stage i. As above, we conclude that m' or a previous message from p votes against S at stage i unless a previous message from p votes for S at stage i. Since m votes for S at stage i, we have a contradiction by Lemma 5.2. ▮

PROPOSITION 5.3. *Let p and q be non-Byzantine processes and let S and S' denote candidate sets for the lth extension of the total order such that there exist messages $s \in S$, $s \notin S'$ and $s' \in S'$, $s' \notin S$. If p decides for S for the lth extension of the total order, then q does not decide for S' for the lth extension.*

Proof. If process p decides for S at stage i, then p determines that at least N_d messages vote for S at stage i and, thus, p has obtained a prefix of the Byzantine causal order containing those messages and all of the messages that they follow. Let m denote any message that votes for S at stage i. By Lemma 5.6, m or a previous message from its source votes against S' at stage i. Thus, p determines that at least N_d messages vote against S' at stage i and, therefore, p decides against S' at stage i. By Proposition 5.2, q does not decide for S'. ▮

The main theorem, Theorem 5.1, guarantees that the total orders determined by non-Byzantine processes are consistent.

THEOREM 5.1. *Let p and q be non-Byzantine processes. If p determines that m is the ith message in the total order, then q does not determine that m' is the ith message, where $m' \neq m$.*

Proof. If p determines that m is the ith message in the total order, then there exists an $l > 0$ such that m is in p's candidate set S for the lth extension of the total order. By Proposition 5.2, if p decides for S, then q does not decide against S and, by Propositions 5.1 and 5.3, q does not decide for S', where $S' \neq S$. Thus, if q selects a candidate set for the lth extension, then q selects S and determines that m is the ith message in the total order since the elements of S are ordered deterministically. ▮

Theorem 5.2 guarantees that no message that follows a message in the Byzantine causal order precedes that message in the total order. Thus, the total order is not arbitrary but is consistent with the Byzantine causal order.

THEOREM 5.2. *If m' follows m in the total order prefix of a non-Byzantine process p, where $m' \neq m$, then m does not follow m' in the Byzantine causal order.*

Proof. Before process p can advance m to its total order prefix, m must become a candidate message. As a candidate message, m only follows messages in the Byzantine causal order that are already in p's total order prefix. Since m' follows m in the total order, m' is not in process p's total order prefix when m becomes a candidate message. Thus, m does not follow m' in the Byzantine causal order. ∎

5.2.2. Probabilistic Termination

The proof of probabilistic termination depends on a specific pattern of communication between nonfaulty processes, called a deciding pattern, defined below. The proof demonstrates that if a nonfaulty process's partial order prefix contains a deciding pattern, then that process decides on the next extension of the total order. The proof also demonstrates that the probability that the partial order prefix contains a deciding pattern increases asymptotically to unity as the size of the prefix increases. There is, however, a small probability that no decision can be made, as required by the impossibility result [FLP].

A *deciding pattern* consists of three *ranks*, as shown in Fig. 3, such that

1. each rank contains messages from $n-k_c-k_b$ distinct nonfaulty processes, and all ranks contain messages from the same $n-k_c-k_b$ processes,

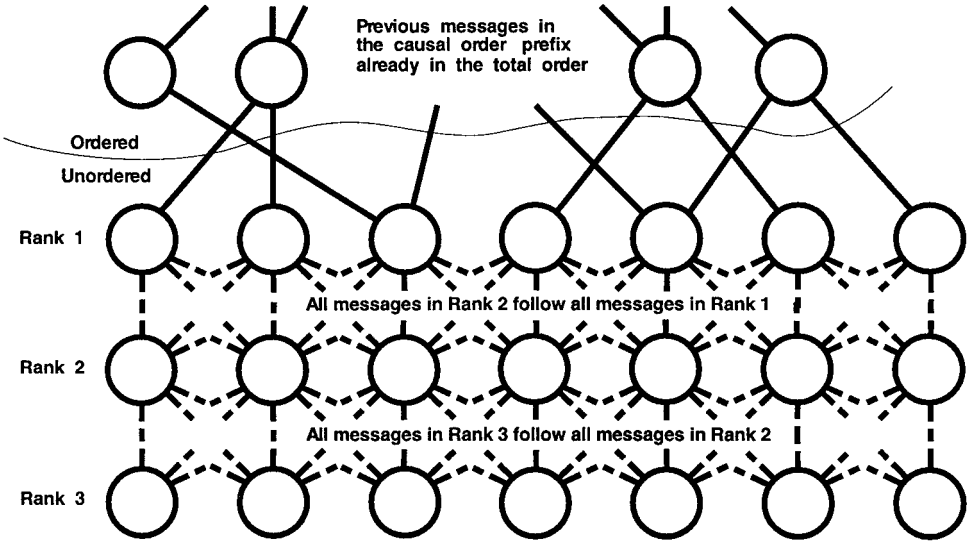

FIG. 3. A deciding pattern of three ranks. Each rank contains messages from $n-k_c-k_b$ distinct nonfaulty processes, and all three ranks contain messages from the same $n-k_c-k_b$ processes. Here $n=9$, $k_c=1$ and $k_b=1$.

2. each message in the second rank directly follows every message in the first rank and no other message, and

3. each message in the third rank directly follows every message in the second rank and no other message.

A deciding pattern is only one of many possible Byzantine causal order patterns that lead to a decision to extend the total order. This particular pattern was chosen because it is easy to analyze, because every message in the second rank follows exactly the same set of messages, aside from itself, and every message in the third rank follows exactly the same set of messages, aside from itself. The nonzero probability of this pattern follows from the admissibility requirements of Section 3. This pattern is, however, quite improbable. Other causal order patterns exist that are much more likely and that lead to a decision to extend the total order just as quickly but that require more complex proofs.

In Proposition 5.6 we show that there always exists a candidate set that can be advanced to the total order at the lth extension; i.e., a non-Byzantine process cannot decide against all available candidate sets.

PROPOSITION 5.6. *Let S^* be the largest set of candidate messages for the lth extension of the total order, each of which is followed by a message from a nonfaulty process. Then $S^* \neq \phi$ and a nonfaulty process p cannot decide against S^*, unless p has decided for a proper subset of S^*.*

Proof. For each nonfaulty process q consider the message m_{qj} in the Byzantine causal order with the smallest sequence number j that is not in process p's $(l-1)$th total order prefix. There exist at least $n-k_c-k_b$ such messages from nonfaulty processes. Furthermore, there exists at least one such message that does not follow any messages other than those in process p's $(l-1)$th total order prefix. Thus, $S^* \neq \phi$.

If p decides against S^* at stage i and p has not decided for a proper subset of S^*, then p determines that at least N_d messages vote against S^* at stage i. Now $N_d = (n+k_c+3k_b+1)/2 > k_c+k_b$ and, therefore, at least one of those messages is from a nonfaulty process. Let m be such a message. By Lemma 5.4, m follows a message m' that votes against S^* at stage 0. Thus, m' follows a candidate message s such that $s \notin S^*$. But, m also follows s, which contradicts the definition of S^*. ∎

The next three lemmas are used to prove Lemma 5.10, which states that there exists a largest stage i at which a message m or a previous message from its source votes on a candidate set S.

LEMMA 5.7. *If message m votes on a candidate set S at stage i, where $i > 0$, then m follows a message that votes on S at stage $i-1$ but does not vote on S at stage i.*

Proof. There are two cases to consider: (1) Message m is originated by a non-Byzantine process. Since m votes on S at stage i, by the Voting criteria, m follows at least two messages that vote on S at stage $i-1$. Consider the set of all messages that m follows and that vote on S at stage $i-1$. This set is finite. Since m is originated by a non-Byzantine process, m does not follow a message that occurs within a nontrivial cycle; thus, no message in this set occurs within a nontrival

cycle. Consequently, this set contains a message m' that follows no message that votes on S at stage $i-1$, aside from itself. Therefore, m' does not vote on S at stage i.

(2) Message m is originated by a Byzantine process. Since m votes on S at stage i and since $N_v = (n - k_c - k_b)/2 > k_b$, m follows at least one message m' from a non-Byzantine process. If m' does not vote on S at stage i, then we are done. If m' votes on S at stage i, then we apply Case (1) to m'. ▮

LEMMA 5.8. *If message m from process p votes on a candidate set S at stage j then, for each i such that* $0 \leqslant i \leqslant j$, *m or a previous message from p votes on S at stage i.*

Proof. The proof is by induction on j. If $j = 0$, then the statement holds trivially. If $j = 1$, then m follows at least $N_v > 0$ messages that vote for S at stage 0 or at least $N_v > 0$ messages that vote against S at stage 0. Each of these messages follows every message in S or a candidate message not in S. Thus, m follows every message in S or a candidate message not in S. Consequently, m or a previous message from p votes on S at stage 0.

Now assume that the statement holds for $j-1$, where $j > 1$. If m votes on S at stage j, then m follows at least $N_v > 0$ messages that vote for S at stage $j-1$ or at least $N_v > 0$ messages that vote against S at stage $j-1$. Each of these messages follows at least two messages that vote on S at stage $j-2$ and at least N_v messages that vote for S at stage $j-2$ or at least N_v messages that vote against S at stage $j-2$. Thus, m follows at least two messages that vote on S at stage $j-2$ and at least N_v messages that vote for S at stage $j-2$ or at least N_v messages that vote against S at stage $j-2$. Consequently, m or a previous message from p votes on S at stage $j-1$. The inductive assumption now yields the result. ▮

LEMMA 5.9. *If message m votes on a candidate set S at stage i, then m follows at least i distinct messages that vote on S at stages* 0 *through* $i-1$.

Proof. The proof is by a simple induction on i using Lemmas 5.7 and 5.8. ▮

LEMMA 5.10. *If message m from process q follows each message in a candidate set S, then there exists an i such that m or a previous message from q votes on S at stage i and neither m nor any previous message from q votes on S at stage* $i+1$.

Proof. The number of messages that precede message m in the Byzantine causal order is finite, say x, and the number of messages that precede any previous message from process q is less than x. By Lemma 5.9, there exists a j (for example, $j = x+1$) such that neither m nor any previous message from q votes on S at stage j. Since m follows each message in S, m or a previous message from q votes on S at stage 0. Thus, there exists an i, $0 \leqslant i < j$, such that m or a previous message from q votes on S at stage i and neither m nor a previous message from q votes on S at stage $i+1$. ▮

Lemma 5.10 is used in the proof of Lemma 5.11, which demonstrates that a deciding pattern is indeed a pattern that enables a nonfaulty process to decide.

LEMMA 5.11. *If the Byzantine causal order prefix of a nonfaulty process p contains a deciding pattern such that each message in its first rank follows all of the messages in a candidate set S, then p decides on S.*

Proof. Every message in the second rank of the deciding pattern follows exactly the same set of messages, aside from itself. Thus, if any message in the second rank or a previous message from its source votes on S at stage i, then every message in the second rank or a previous message from its source votes on S at stage i. Consequently, by Lemma 5.10, there exists a largest stage i at which every message in the second rank or a previous message from its source votes on S.

Now every message in the third rank follows the same set of messages, aside from itself. In particular, each of these messages follows exactly the same set of at least $n-k_c-k_b$ messages that vote on S at stage i and follows no message that votes on S at stage $i+1$, aside from itself. Consequently, by the Voting criteria, all $n-k_c-k_b$ third-rank messages vote for S at stage $i+1$ or all vote against S at stage $i+1$. Since $N_d \leqslant n-k_c-k_b$, by the Decision criteria and induction on the cardinality of the candidate sets, process p decides on S at stage $i+1$. ▮

Lemma 5.12 shows that the probability that a Byzantine causal order prefix contains a deciding pattern that follows a candidate set increases as the size of the prefix increases, while Lemma 5.13 shows that the probability that a nonfaulty process selects a candidate set for the lth extension increases as the number of steps taken by the process increases.

LEMMA 5.12. *Let S be a candidate set, each message of which is followed by a message from a nonfaulty process. The probability that a Byzantine causal order prefix of size x obtained by a nonfaulty process p contains a deciding pattern, such that each message in its first rank follows all of the messages in S, increases asymptotically to unity as x tends to infinity.*

Proof. Consider a substructure of process p's Byzantine causal order prefix consisting of three consecutive messages from each of $n-k_c-k_b$ nonfaulty processes, all of which follow the messages in the candidate set S. A message m in the first rank of a deciding pattern, directly followed by a message in the second rank, is an event $\mathrm{E}(m, p, q)$ and, similarly, for the second and third ranks.

By Fairness property 1, for each such substructure, the probability of each such event $\mathrm{E}(m, p, q)$ is greater than some positive constant ϵ, while by Fairness property 2, these probabilities are independent. Thus, the probability that the substructure is a deciding pattern is greater than some positive constant δ.

By Fairness property 2, each increment to the Byzantine causal order prefix constitutes a Bernoulli trial, with a probability greater than δ that the prefix contains a substructure that is a deciding pattern. As x tends to infinity, the number of messages added to the Byzantine causal order prefix and, thus, the number of such Bernoulli trials tends to infinity.

Consequently, the probability that the Byzantine causal order prefix contains a deciding pattern, such that each message in its first rank follows all messages in the candidate set S, increases asymptotically to unity as x tends to infinity. ▮

LEMMA 5.13. *The probability that a nonfaulty process p selects a set S of candidate messages for the lth extension of its total order prefix increases asymptotically to unity as the number t of steps taken by p tends to infinity.*

Proof. Let S^* be the largest set of candidate messages in the Byzantine causal order for the lth extension of the total order, each of which is followed by a message from a nonfaulty process. By Proposition 5.6, p cannot decide against S^* unless p has decided for a proper subset of S^*. By Lemma 5.11, if all of the messages of S^* are followed by each of the messages of a deciding pattern, then p decides for S^* or for a proper subset of S^*. By Lemma 5.12, the probability that each message of S^* is followed by all of the messages of a deciding pattern tends to unity as the size x of p's Byzantine causal order prefix tends to infinity and, thus, as the number t of steps taken by p tends to infinity. ▌

This leads directly to Theorems 5.3 and 5.4, which establish the Probabilistic Termination requirements.

THEOREM 5.3. *The probability that a nonfaulty process p places an ith message in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. By Lemma 5.13 and elementary probability theory, for any given l, the probability that a nonfaulty process p selects a set S^l of candidate messages for the lth extension of its total order prefix increases asymptotically to unity as the number of steps taken by p tends to infinity. Choose an l such that $l \geqslant i$. Then the ith message in the total order is contained in the lth total order prefix. ▌

THEOREM 5.21. *For each message m from a nonfaulty process q, the probability that a nonfaulty process p places message m in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. Since message m is broadcast by a nonfaulty process q, m is followed by a message from each nonfaulty process. Since each such message follows a finite number of messages in the Byzantine causal order, there exists a finite number, say l, of messages that are followed by a message from a nonfaulty process and that do not follow m. By Lemma 5.13, the probability that p selects a set S^l of candidate messages for the lth extension of its total order prefix increases asymptotically to unity as the number of steps taken by p tends to infinity. At the lth extension, either m is already in its $(l-1)$th total order prefix or all messages that are followed by a message from a nonfaulty process and that do not follow m are in its $(l-1)$th total order prefix. In the latter case, m is the only candidate message and is selected for the lth extension. ▌

6. THE $3k_c + 3k_b$ TOTAL ALGORITHM

For this algorithm, we assume that the relationship between the number n of processes, the number k_c of crash processes and the number k_b of Byzantine processes is $3k_c + 3k_b < n$. The Eligibility criteria now has a third requirement. This third requirement essentially converts the messages from Byzantine processes into

messages from crash processes. With this requirement, mutants that are not followed by N_e messages never become eligible to vote. The rest of the algorithm is the same as the $3k_c + 5k_b$ algorithm with k_b set to 0 and k_c set to $k_c + k_b$ and by removing all references to Byzantine processes.

The number of messages required for eligibility to vote, the number of votes required for a further vote and the number of votes required for a decision are at least N_e, N_v, and N_d, where

$$N_e = (n + k_b + 1)/2, \qquad N_v = (n - k_c - k_b)/2, \qquad N_d = (n + k_c + k_b + 1)/2.$$

Eligibility Criteria. At stage i, where $i \geqslant 0$,

- A non-Byzantine process determines that a message m is eligible to vote on S if

  — the process has obtained a prefix A of the Byzantine causal order such that $m \in A$,

  — no previous message from the source of m votes on S at stage i, and

  — the prefix A contains at least N_e messages from distinct processes, each of which follows m and does not follow a mutant of m.

The proof for the $3k_c + 3k_b$ algorithm is based on the following lemmas. The rest of the proof can be obtained from that for the $3k_c + 5k_b$ algorithm by setting k_b to 0 and k_c to $k_c + k_b$ and by removing all references to Byzantine processes.

LEMMA 6.1. *If message m is eligible to vote on S at stage i, then no mutant m' of m is eligible to vote on S at stage i.*

Proof. If m is eligible to vote on S at stage i, then m is followed by $N \geqslant N_e = (n + k_b + 1)/2$ messages from distinct processes, none of which follows a mutant m'. Similarly, if the mutant m' of m is eligible to vote on S at stage i, then m' is followed by $N' \geqslant N_e = (n + k_b + 1)/2$ messages from distinct processes, none of which follows the message m. At most k_b of the processes are Byzantine and, thus, each can generate two messages, one following m but not m' and the other following m' but not m. Thus, the number of processes is at least $N + N' - k_b \geqslant (n + k_b + 1)/2 + (n + k_b + 1)/2 - k_b = n + 1$, which is a contradiction. ▮

LEMMA 6.2. *If message m has no mutant, then m is eligible to vote on S at stage i.*

Proof. Messages from at most k_b Byzantine processes may claim to follow a mutant of m but, since m has no mutant, m is followed by messages from at least $n - k_c - k_b$ nonfaulty processes that do not claim to follow a mutant of m. Since $N_e = (n + k_b + 1)/2 \leqslant n - k_c - k_b$, m is eligible to vote on S at stage i. ▮

7. THE $2k_c + 5k_b$ TOTAL ALGORITHM

For this algorithm, we assume that the number n of processes, the number k_c of crash processes, and the number k_b of Byzantine processes satisfies the constraint

$2k_c + 5k_b < n$. The algorithm is defined by the Eligibility, Voting, Proposing, and Decision criteria given below; these criteria determine whether the candidate set S is chosen for inclusion in the total order.

The number of messages related to an indifferent proposal, the number of votes required for a proposal and the number of proposals required for a decision are at least N_v, N_p, and N_d, respectively, where

$$N_v = n - k_c - k_b, \qquad N_p = (n + k_b + 1)/2, \qquad N_d = k_c + 2k_b + 1.$$

Eligibility Criteria. At stage i, where $i \geqslant 0$,

- a non-Byzantine process determines that a message m is eligible to vote on S if
  - — the process has obtained a prefix A of the Byzantine causal order such that $m \in A$, and
  - — no previous message from the source of m has voted on S at stage i;
- a non-Byzantine process determines that a message m is eligible to propose on S if
  - — the process has obtained a prefix A of the Byzantine causal order such that $m \in A$, and
  - — no previous message from the source of m has proposed on S at stage i.

Voting Criteria. At stage 0,

- a message votes for S if it follows every message in S and it follows no other candidate message (a candidate message votes for the set containing only itself);
- a message votes against S if it follows a candidate message not in S (a candidate message votes against all sets of which it is not a member).

At stage i, where $i > 0$,

- a message votes for S if
  - — it follows a message that proposes for S at stage $i-1$;
- a message votes against S if
  - — it follows a message that proposes against S at stage $i-1$,

or

  - — it follows no message that proposes for or against S at stage $i-1$, and
  - — it follows at least N_v messages that propose indifferent to S at stage $i-1$.

Proposing Criteria. At stage i, where $i \geqslant 0$,

- a message proposes for S if
  - — it follows at least N_p messages that vote for S at stage i;
- a message proposes against S if
  - — it follows at least N_p messages that vote against S at stage i;

- a message proposes indifferent to S if
  - — it does not propose for or against S at stage i, and
  - — it follows at least N_v messages that vote on S at stage i.

Decision Criteria. At stage i, where $i \geqslant 0$,

- a non-Byzantine process decides for S if
  - — it determines that at least N_d messages propose for S at stage i, and
  - — for each proper subset of S, it decides against that proper subset;
- a non-Byzantine process decides against S if
  - — it determines that at least N_d messages propose against S at stage i, or
  - — it decides for a proper subset of S.

The values of N_v, N_p, and N_d and the constraint $2k_c + 5k_b < n$ are derived from the following properties: Votes and decisions do not conflict ($N_v + N_d > n + k_b$), proposals do not conflict ($N_p + N_p > n + k_b$), decisions do not conflict ($N_d > k_c + k_b$), stages of voting and proposing advance ($N_v \leqslant n - k_c - k_b$), proposals for or against are feasible ($N_p \leqslant n - k_c - k_b$ and $N_p \leqslant N_v - k_b$), and decisions are feasible ($N_d \leqslant n - k_c - k_b$).

The proof of correctness for the $2k_c + 5k_b$ algorithm is similar to that for the $3k_c + 5k_b$ algorithm and can be found in Appendix I.

8. THE $2k_c + 3k_b$ TOTAL ALGORITHM

For this algorithm we assume that the number n of processes, the number k_c of crash processes, and the number k_b of Byzantine processes satisfies the constraint $2k_c + 3k_b < n$. The Eligibility criteria for voting are the same as those for the $3k_c + 3k_b$ algorithm. In addition, here we have identical Eligibility criteria for proposing. The rest of the algorithm is the same as the $2k_c + 5k_b$ algorithm with k_b set to 0 and k_c set to $k_c + k_b$ and by removing all references to Byzantine processes.

The number of messages required for eligibility to vote, the number of messages related to an indifferent proposal, the number of votes required for a proposal, and the number of proposals required for a decision are at least N_e, N_v, N_p, and N_d, where

$$N_e = (n + k_b + 1)/2, \qquad N_v = n - k_c - k_b, \qquad N_p = (n+1)/2, \qquad N_d = k_c + k_b + 1$$

Eligibility Criteria. At stage i, where $i \geqslant 0$,

- a non-Byzantine process determines that a message m is eligible to vote (propose) on S if
  - — the process has obtained a prefix A of the Byzantine causal order such that $m \in A$,

— no previous message from the source of m votes (proposes) on S at stage i, and

— the prefix A contains at least N_e messages from distinct processes, each of which follows m and does not follow a mutant of m.

The proof for the $2k_c + 3k_b$ algorithm is based on the following lemmas. The rest of the proof can be obtained from that for the $2k_c + 5k_b$ algorithm by setting k_b to 0 and k_c to $k_c + k_b$ and by removing all references to Byzantine processes. Since $2(k_c + k_b) < 2k_c + 3k_b < n$, our assumption $2k_c + 3k_b < n$ for this algorithm implies the assumption $2(k_c + k_b) < n$ for the $2k_c + 5k_b$ algorithm, where there are $k = k_c + k_b$ crash processes and no Byzantine processes.

The proof of correctness is similar to that for the $3k_c + 3k_b$ algorithm and can be found in Appendix II.

9. PERFORMANCE TRADE-OFFS OF THE ALGORITHMS

The primary performance measure for a total ordering algorithm, such as one of the Total algorithms, is the mean latency from transmission of a message to delivery of that message into the total order. The worst-case latency is, unfortunately, not a useful measure since it is necessarily infinite. The analysis of the latency of a total ordering algorithm in the presence of Byzantine faults, or even of message loss, is difficult. Fortunately, in a well-designed system that uses LANs or ATM connections, the probability of message loss is low ($< 10^{-6}$) and the probability of crash faults is even lower ($< 10^{-10}$), and neither significantly affects the mean latency. Moreover, Byzantine faults can be assumed to be rare, since a Byzantine fault leads to the identification and removal of the faulty process.

Consequently, we provide here a straightforward analysis of the mean latency for each of the Total algorithms in the most probable case that for each message in the causal order the probability that a process originated that message is the same for all processes, each message is received by all processes, there are no faulty processes, and the causal order is a linear order. Under low or moderate load and assuming reliable message communication, a process can have received all prior messages from other processes and can have prepared the appropriate acknowledgments before transmitting its own messages, resulting in a linear causal order. Under high load, two or more processes can transmit simultaneously or almost simultaneously, resulting in concurrent messages in the causal order and a more difficult latency analysis, and the latency can be affected by contention for the communication medium itself. The analysis of latency under high load is the subject of ongoing research.

Under the assumptions given above, there is never more than one candidate message and all messages vote for that candidate message. The mean latency is the mean number of messages that must be considered in the causal order to obtain the required number of votes from distinct processes. Thus, the mean number of messages required to obtain messages from m distinct processes for a system of n processes is

$$
\begin{aligned}
\mathrm{L}(n, m) &= 1 + \frac{n-1}{n}\left(1 + 2\left(\frac{1}{n}\right) + 3\left(\frac{1}{n}\right)^2 + \cdots\right) + \frac{n-2}{n}\left(1 + 2\left(\frac{2}{n}\right) + 3\left(\frac{2}{n}\right)^2 + \cdots\right) \\
&\quad + \cdots + \frac{n-m+1}{n}\left(1 + 2\left(\frac{m-1}{n}\right) + 3\left(\frac{m-1}{n}\right)^2 + \cdots\right) \\
&= 1 + \frac{n-1}{n}\left(\frac{n}{n-1}\right)^2 + \frac{n-2}{n}\left(\frac{n}{n-2}\right)^2 + \cdots + \frac{n-m+1}{n}\left(\frac{n}{n-m+1}\right)^2 \\
&= 1 + \frac{n}{n-1} + \frac{n}{n-2} + \cdots + \frac{n}{n-m+1} \\
&= \sum_{i=0}^{m-1} \frac{n}{n-i}.
\end{aligned}
$$

The $3k_c + 5k_b$ algorithm requires only N_d votes from distinct processes to decide. Thus, the mean latency is $\mathrm{L}(n, N_d)$. The $3k_c + 3k_b$ algorithm has the extra eligibility requirement that a message must be acknowledged by N_e messages before it can become eligible to vote. Thus, the mean latency is $\mathrm{L}(n, N_e) + \mathrm{L}(n, N_d) - 1$.

The $2k_c + 5k_b$ algorithm involves proposals in addition to votes. Since a message counts as both the last required message to vote and also as the first message to propose, the mean latency is $\mathrm{L}(n, N_p) + \mathrm{L}(n, N_d) - 1$. The $2k_c + 3k_b$ algorithm also has the extra eligibility requirement. Thus, the mean latency is $\mathrm{L}(n, N_e) + \mathrm{L}(n, N_p) + \mathrm{L}(n, N_d) - 2$.

Figure 4 shows the mean latencies for the $3k_c + 5k_b$ algorithm and for the $3k_c + 3k_b$ algorithm. The latencies are shown for various numbers of processes and for various levels of resilience. In representing the resilience, the number of crash faults is indicated before the number of Byzantine faults; thus, 3/2 represents a system resilient to three crash faults and two Byzantine faults. Note that in some cases the latencies are the same for two different resiliencies. An odd–even effect is clearly visible in the graphs.

As the graphs show, the mean latency is lower for the $3k_c + 5k_b$ algorithm than for the $3k_c + 3k_b$ algorithm, but there are many cases in which, for the same number of processes, the $3k_c + 3k_b$ algorithm can tolerate a larger number of faults. The increased resilience to Byzantine faults of the $3k_c + 3k_b$ algorithm has a price; the extra eligibility requirement increases the latency. For example, in a 12-process system resilient to two crash faults and one Byzantine fault, the mean latency is 15.24 messages for the $3k_c + 5k_b$ algorithm and 21.08 messages for the $3k_c + 3k_b$ algorithm. In a 10-process system, however, resilience to two crash faults and one Byzantine fault is impossible for the $3k_c + 5k_b$ algorithm, but it can be achieved for the $3k_c + 3k_b$ algorithm. Thus, neither algorithm dominates the other.

Figure 5 shows the corresponding mean latencies for the $2k_c + 5k_b$ algorithm and for the $2k_c + 3k_b$ algorithm. It is evident that the differences between these two algorithms are very similar to those between the first two algorithms. Again, neither algorithm dominates the other.

When comparing Fig 4 with Fig. 5, however, it is much less clear whether the $3k_c + 5k_b$ algorithm has a lower latency than the $2k_c + 5k_b$ algorithm, or whether

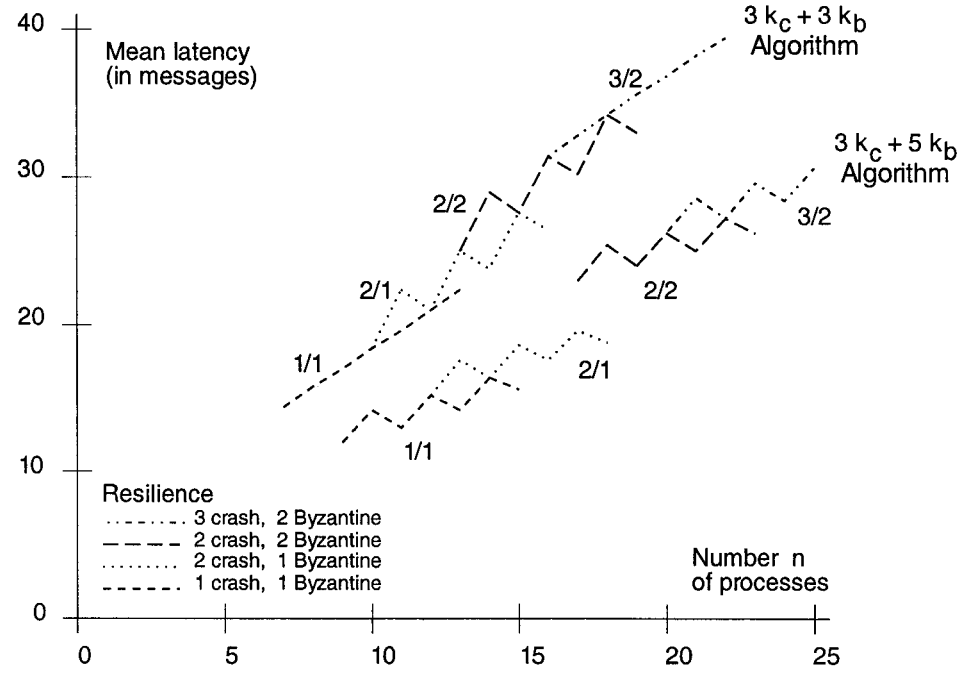

FIG. 4. The mean latency, measured in messages, from transmission to delivery into the total order for various numbers of processes and for various levels of resilience. The resilience is indicated as the number of crash faults followed by the number of Byzantine faults.

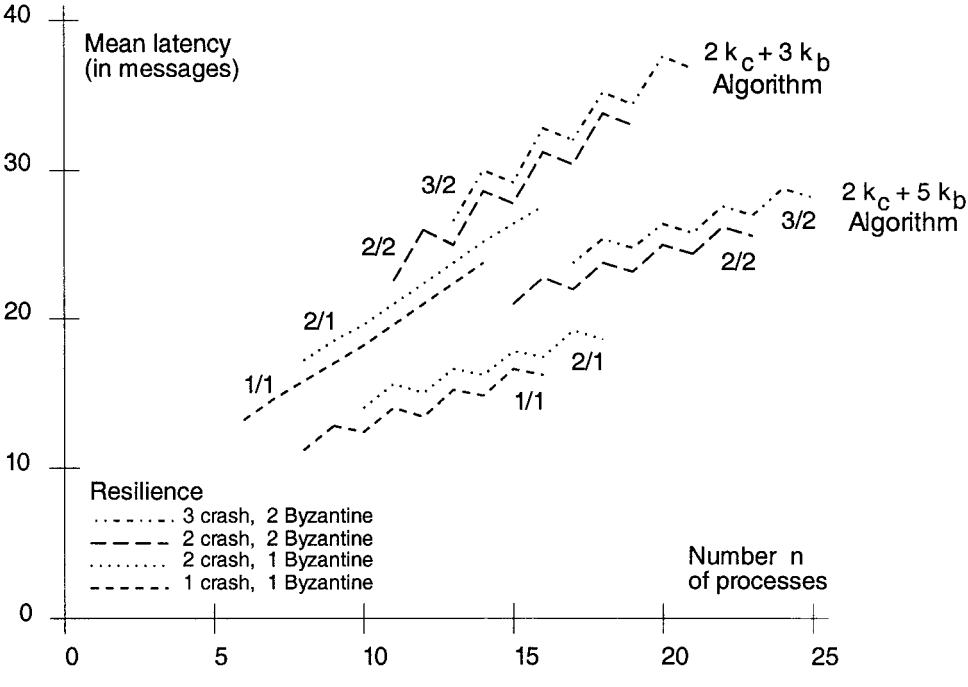

FIG. 5. The mean latency, measured in messages, from transmission to delivery into the total order for various numbers of processes and for various levels of resilience. The resilience is indicated as the number of crash faults followed by the number of Byzantine faults.

the $3k_c + 3k_b$ algorithm has a lower latency than the $2k_c + 3k_b$ algorithm. In each case, for some numbers of processes and levels of resilience, one has a lower latency and, for other numbers of processes and levels of resilience, the other has a lower latency. Overall, none of the four algorithms dominates any of the other algorithms and the choice of algorithm must be made for each application individually, based on the specific needs of that application.

9.1. Mixed-Mode Operation

To illustrate mixed-mode operation of the Total algorithms, we consider the $3k_c + 5k_b$ algorithm and a system of $n = 24$ processes, with $k_c = 2$ and $k_b = 3$. For these values of k_c and k_b, $N_v = 9.5$ and $N_d = 18$. In this case, the algorithm can tolerate two crash faults and three Byzantine faults, or three crash faults and two Byzantine faults, or four crash faults and one Byzantine fault, or five crash faults and no Byzantine faults, since any crash fault can be covered by a Byzantine fault. Examination of the inequalities at the end of Section 4 reveals that, with $N_v = 9.5$ and $N_d = 18$, the algorithm can also tolerate six crash faults and no Byzantine faults. Thus, a reduction in the number of Byzantine faults allows an increase in the total number of faults that can be tolerated. The mean latency for this case is $L(24, 18) = 31.8$ messages.

If, instead, we consider the $3k_c + 5k_b$ algorithm and a system of $n = 24$ processes, with $k_c = 6$ and $k_b = 1$, then we obtain $N_v = 8.5$ and $N_d = 17$. In this case, the algorithm can tolerate six crash faults and one Byzantine fault, or seven crash faults and no Byzantine faults. The mean latency for this case is $L(24, 17) = 21.4$ messages. Note that in the previous case the ability to tolerate more Byzantine faults reduced the total number of faults that could be tolerated, and also increased the latency.

10. CONCLUSION

The four Total algorithms presented here derive a total order from a causal order on messages in the presence of crash and Byzantine faults. These algorithms are distributed algorithms that are executed independently at each of the processes and require no additional communication between processes. The algorithms employ a multistage voting strategy to achieve agreement on the total order and exploit the random structure of the causal order to achieve probabilistic termination. They ensure that nonfaulty processes construct identical total orders on messages and that non-Byzantine processes construct consistent total orders, provided that the resilience requirements are satisfied.

Two of the four Total algorithms include messages from Byzantine processes in the voting. These two algorithms exhibit excellent latency, but do not achieve the maximum possible resilience. The other two Total algorithms use an extra eligibility requirement for voting that essentially converts Byzantine faults into crash faults and increases the resilience of the algorithms. Converting Byzantine faults into crash faults seldom yields the most efficient algorithms. We continue to seek a voting strategy that tolerates the theoretical maximum number of Byzantine faults and that matches the lower latency of our most efficient algorithms.

APPENDIX I

We present here the proof of correctness for the $2k_c + 5k_b$ algorithm given in Section 7. This proof is similar to that for the $3k_c + 5k_b$ algorithm.

I.1. Partial Correctness

The proof of partial correctness involves showing that if two non-Byzantine processes p and q make lth extensions to their total order prefixes, then the candidate sets they choose for those extensions are identical and, thus, their lth total order prefixes are identical. The proof is by induction on l. The base case is trivial since, if $l = 0$, then all of these sets are empty. The heart of the proof constitutes establishing the inductive step.

As in the $3k_c + 5k_b$ algorithm, we have the following lemma which states that the only information used by the algorithm is the Byzantine causal order on messages.

LEMMA 7.1. *Let p and q be non-Byzantine processes. If p and q each determine the vote of a message m on a candidate set S at stage i, then they determine the same vote of m on S at stage i.*

Proof. The proof is similar to that of Lemma 5.1. ▮

LEMMA 7.2. *Each non-Byzantine process p broadcasts at most one message that votes on a candidate set S at stage i; that message does not vote both for and against the candidate set S at stage i.*

Proof. Since p broadcasts at most one message that votes on S at stage i, it suffices to show that if a message m votes for (against) a candidate set S at stage i, then m does not vote against (for) S at stage i. The proof is by induction on i. For $i = 0$, the Voting criteria imply the statement.

We now assume the statement for $i-1$ and argue by contradiction to establish the statement for i. Thus, we assume that m votes both for and against S at stage i. By the Voting criteria, since m votes for S at stage i, it follows a message that proposes for S at stage $i-1$. Consequently, since m votes against S at stage i, it follows a message that proposes against S at stage $i-1$. By the Proposing criteria, m follows $N \geqslant N_p$ messages that vote against S at stage $i-1$. Moreover, the inductive assumption implies that those messages do not vote for S at stage $i-1$. Thus, since a Byzantine process can broadcast messages that vote for and against S at stage $i-1$, the number of messages from distinct processes that vote for S at stage $i-1$ is at most $n - N + k_b \leqslant n - N_p + k_b = (n - k_b - 1)/2 + k_b < (n + k_b + 1)/2 + k_b = N_p + k_b$. Consequently, no message proposes for S at stage $i-1$, which is a contradiction. ▮

LEMMA 7.2.1. *If message m votes against (for) a candidate set S at stage i, then m follows a message that votes against (for) S at stage $i-1$, where $i > 0$.*

Proof. Assume that m votes against S at stage i and that m follows no message that votes against S at stage $i-1$. Then, by the Proposing criteria, m follows no message that proposes against S at stage $i-1$. But m votes against S at stage i.

Therefore, by the Voting criteria, m follows at least N_v messages that propose indifferent to S at stage $i-1$. Hence, by the Proposing criteria, m follows $N \geqslant N_v$ messages that vote on S at stage $i-1$. If all such messages from non-Byzantine processes vote for S, then m proposes for S at stage $i-1$ (rather than proposing indifferent), since $2k_c + 5k_b < n$ implies that $N_v = n - k_c - k_b \geqslant (n + k_b + 1)/2 + k_b = N_p + k_b$. Thus, m votes for S at stage i, which is a contradiction. Consequently, at least one of the N messages votes against S at stage $i-1$. The proof in the case that m votes for S at stage i is obvious. ▮

LEMMA 7.2.2. *If message m proposes for (against) a candidate set S at stage i, then no message proposes against (for) S at stage i.*

Proof. The statement follows from Lemma 7.2 and the fact that $N_p + N_p > n + k_b$ since only a Byzantine process can broadcast messages that propose for and against S at stage i. ▮

LEMMA 7.2.3. *Each non-Byzantine process p broadcasts at most one message that proposes on a candidate set S at stage i; that message proposes only once at stage i.*

Proof. The first statement follows from the requirement that a message proposes on a candidate set S at stage i only if no previous message from its source has proposed on S at stage i. The second statement is given by Lemma 7.2.2. ▮

LEMMA 7.3. *If message m follows at least N_d messages that propose for (against) a candidate set S at stage $i-1$, then no message m' votes against (for) S at stage i, where $i > 0$.*

Proof. Assume that message m' votes against S at stage i. Then, either (1) m' follows a message that proposes against S at stage $i-1$, or (2) m' follows at least N_v messages that propose indifferent to S at stage $i-1$. By the hypothesis and Lemma 7.2.2, Case (1) cannot occur. In Case (2), by the hypothesis, Lemma 7.2.3, and the fact that a Byzantine process can broadcast messages that propose for (against) S and indifferent to S at stage $i-1$, at most $n - N_d + k_b = n - k_c - 2k_b - 1 + k_b < n - k_c - k_b = N_v$ messages propose indifferent to S at stage $i-1$, which is a contradiction. The proof for the alternative statement of the lemma is similar but simpler, because only Case (1) arises. ▮

LEMMA 7.4. *If message m votes against (for) a candidate set S at stage j then, for each i such that $0 \leqslant i \leqslant j$, m follows a message that votes against (for) S at stage i.*

Proof. The proof is similar to that of Lemma 5.4 and uses Lemma 7.2.1 in place of the Voting criteria. ▮

LEMMA 7.5. *If no message from a non-Byzantine process votes against (for) a candidate set S at stage i, then no message from any process votes against (for) S at stage j, where $j \geqslant i$. Likewise, if some process p decides for (against) a candidate set S at stage i, then no message proposes against (for) S at stage j, where $j > i$.*

Proof. The proof of the first statement is identical to that of the first statement of Lemma 5.5. To prove the second statement, we note that if p decides for a

candidate set S at stage i, then p determines that at least N_d messages propose for S at stage i. By Lemma 7.3, no message votes against S at stage $i+1$. By the first statement, no message votes against S at stage j, where $j \geqslant i+1$. Therefore, by the Proposing criteria, no message proposes against S at stage j, where $j > i$. ▮

PROPOSITION 7.1. *Let p and q be non-Byzantine processes. If p decides for a candidate set S for the lth extension of the total order, then q does not decide for a proper subset S' of S for the lth extension.*

Proof. The proof is similar to that of Proposition 5.1, but it is based on Lemma 7.1, Lemma 7.2.2, and Lemma 7.5. ▮

PROPOSITION 7.2. *Let p and q be non-Byzantine processes. If p decides for (against) a candidate set S for the lth extension of the total order, then q does not decide against (for) S for the lth extension.*

Proof. The proof is similar to that of Proposition 5.2, but it is based on Proposition 7.1, Lemma 7.1, Lemma 7.2.2, and Lemma 7.5. ▮

The statement of Lemma 7.6 is weaker than that of Lemma 5.6, but it suffices to prove Lemma 7.6.1, which is needed to prove Proposition 7.3 below.

LEMMA 7.6. *Let S and S' denote candidate sets for the lth extension of the total order such that there exist $s \in S$, $s \notin S'$ and $s' \in S'$, $s' \notin S$. If message m from a non-Byzantine process p votes for S at stage i, then no message from p votes for S' at stage i.*

Proof. The proof is by induction on i. If message m from process p votes for S at stage 0, then it follows the candidate message $s \in S$ and no previous message from p votes on S. Since m follows s and $s \notin S'$, m votes against S' at stage 0. By Lemma 7.2, no message from p votes for S' at stage 0.

We now assume the statement for $i-1$. If m votes for S at stage i then, by the Voting and Proposing criteria, m follows $N \geqslant N_p$ messages from distinct processes that vote for S at stage $i-1$. By the inductive assumption, for each such message from a non-Byzantine process, no message from that process votes for S' at stage $i-1$. Thus, at most $n - N + k_b \leqslant n - N_p + k_b = (n + k_b - 1)/2 < (n + k_b + 1)/2 = N_p$ messages from distinct processes vote for S' at stage $i-1$. Consequently, by the Proposing criteria, no message proposes for S' at stage $i-1$ and, by the Voting criteria, no message votes for S' at stage i. ▮

LEMMA 7.6.1. *Let S and S' denote candidate sets for the lth extension of the total order such that there exist $s \in S$, $s \notin S'$ and $s' \in S'$, $s' \notin S$. If message m proposes for S at stage i, then no message proposes for S' at stage i.*

Proof. If m proposes for S at stage i then, by the Proposing criteria, at least N_p messages from distinct processes vote for S at stage i. By Lemma 7.6, for each such message from a non-Byzantine process, no message from that process votes for S' at stage i. Thus, at most $n - N_p + k_b = (n + k_b - 1)/2 < (n + k_b + 1)/2 = N_p$ messages from distinct processes vote for S' at stage i. By the Proposing Criteria, no message proposes for S' at stage i. ▮

PROPOSITION 7.3. *Let p and q be non-Byzantine processes, and let S and S' denote candidate sets for the lth extension of the total order such that there exist messages $s \in S$, $s \notin S'$ and $s' \in S'$, $s' \notin S$. If p decides for S for the lth extension of the total order, then q does not decide for S' for the lth extension.*

Proof. If p decides for S at stage i then, by the Decision criteria, p determines that a message (in fact, at least N_d messages) proposes for S at stage i. Thus, by Lemma 7.6.1, no message proposes for S' at stage i. Consequently, by the Decision criteria, q does not decide for S' at stage i. ▮

The main theorem, Theorem 7.1, guarantees that the total orders determined by non-Byzantine processes are consistent.

THEOREM 7.1. *Let p and q be non-Byzantine processes. If p determines that m is the ith message in the total order, then q does not determine that m' is the ith message, where $m' \neq m$.*

Proof. The proof is identical to that of Theorem 5.1. ▮

Theorem 7.2 guarantees that no message that follows a message in the Byzantine causal order precedes that message in the total order. Thus, the total order is not arbitrary but is consistent with the Byzantine causal order.

THEOREM 7.2. *If m' follows m in the total order prefix of a non-Byzantine process p, where $m' \neq m$, then m does not follow m' in the Byzantine causal order.*

Proof. The proof is identical to that of Theorem 5.2. ▮

I.2. Probabilistic Termination

To prove probabilistic termination we use the following definition.

A *deciding pattern* consists of five *ranks* such that

1. each rank contains messages from $n - k_c - k_b$ distinct nonfaulty processes, and all ranks contain messages from the same $n - k_c - k_b$ processes,
2. each message in the second rank directly follows every message in the first rank and no other message,
3. each message in the third rank directly follows every message in the second rank and no other message,
4. each message in the fourth rank directly follows every message in the third rank and no other message, and
5. each message in the fifth rank directly follows every message in the fourth rank and no other message.

In Proposition 7.6 we show that a process cannot decide against all available candidate sets.

PROPOSITION 7.6. *Let S^* be the largest set of candidate messages in the Byzantine causal order for the lth extension of the total order, each of which is followed by a message from a nonfaulty process. Then $S^* \neq \phi$ and, if a non-Byzantine process p*

makes the lth extension of its total order prefix, then p cannot decide against S unless p has decided for a proper subset of S*.*

Proof. The proof that $S^* \neq \phi$ is identical to the corresponding proof of Proposition 5.6. Now, if p decides against S^* at stage i and p has not decided for a proper subset of S^*, then p determines that at least N_d messages propose against S^* at stage i. Each of those messages follows at least N_p messages that vote against S^* at stage i. Since $N_p = (n + k_b + 1)/2 > k_c + k_b$ (since $2k_c + 5k_b < n$), at least one of those messages is from a nonfaulty process. Let m be such a message. By Lemma 7.4, m follows a message m' that votes against S^* at stage 0. Thus, m' follows a candidate message s such that $s \notin S^*$. But, m also follows s, which contradicts the definition of S^*. ▮

The next three lemmas are used to prove Lemma 7.10, which states that there exists a largest stage i at which a message m or a previous message from its source votes on a candidate set S.

LEMMA 7.7. *If message m votes on a candidate set S at stage i, where $i > 0$, then m follows a message that votes on S at stage $i-1$ but does not vote on S at stage i.*

Proof. There are two cases to consider: (1) Message m is originated by a non-Byzantine process. Since m votes on S at stage i, by the Voting criteria m follows at least two messages that vote on S at stage $i-1$. Consider the set of all messages that m follows and that vote on S at stage $i-1$. This set is finite. Since m is originated by a non-Byzantine process, m does not follow a message that occurs within a nontrivial cycle. Consequently, this set contains a message m' that follows no message that votes on S at stage $i-1$, aside from itself. Since $N_p = (n + k_b + 1)/2 > 1$ and $N_v = n - k_c - k_b > 1$ (since $2k_c + 5k_b < n$), m' does not propose on S at stage $i-1$ and thus does not vote on S at stage i.

(2) Message m is originated by a Byzantine process. Since m votes on S at stage i and since $N_v = n - k_c - k_b > k_b$, m follows at least one message m' from a non-Byzantine process. If m' does not vote on S at stage i, then we are done. If m' votes on S at stage i, then we apply Case (1) to m'. ▮

LEMMA 7.8. *If message m from process p votes on a candidate set S for the lth extension of the total order at stage j then, for each i such that $0 \leqslant i \leqslant j$, m or a previous message from p votes on S at stage i.*

Proof. The proof is by induction on j. If $j = 0$, then the statement holds trivially. If $j = 1$, then three cases arise: (1) m follows a message m' that proposes for S at stage 0 and, thus, m' follows at least N_p messages that vote for S at stage 0, (2) m follows a message m' that proposes against S at stage 0 and, thus, m' follows at least N_p messages that vote against S at stage 0, or (3) m follows at least N_v messages that propose indifferent to S at stage 0, each of which follows at least N_v messages that vote on S at stage 0. In each case, m follows a message that votes on S at stage 0 and, thus, m follows every message in S or a candidate message not in S. Consequently, m or a previous message from p votes on S at stage 0.

Now assume that the statement holds for $j-1$, where $j>1$. If m votes on S at stage j, then three cases arise: (1) m follows a message m' that proposes for S at stage $j-1$ and, thus, m' follows at least N_p messages that vote for S at stage $j-1$, each of which follows a message that proposes for S at stage $j-2$, (2) m follows a message m' that proposes against S at stage $j-1$ and, thus, m' follows at least N_p messages that vote against S at stage $j-1$, each of which follows a message that proposes against S at stage $j-2$ or follows at least N_v messages that propose indifferent to S at stage $j-2$, or (3) m follows at least N_v messages that propose indifferent to S at stage $j-1$, each of which follows at least N_v messages that vote on S at stage $j-1$; each of these messages either follows a message that proposes for S at stage $j-2$, follows a message that proposes against S at stage $j-2$, or follows at least N_v messages that propose indifferent to S at stage $j-2$. In each case, m or a previous message from p votes on S at stage $j-1$. The inductive assumption now gives the result. ▮

LEMMA 7.9. *If message m votes on a candidate set S for the lth extension of the total order at stage i, then m follows at least i distinct messages that vote on S at stages* 0 *through* $i-1$.

Proof. The proof is identical to that of Lemma 5.9. ▮

LEMMA 7.10. *If message m from process q follows each message in a candidate set S, then there exists an i such that m or a previous message from q votes on S at stage i and neither m nor any previous message from q votes on S at stage* $i+1$.

Proof. The proof is identical to that of Lemma 5.10. ▮

The above lemma is used in the proof of Lemma 7.11, which demonstrates that a deciding pattern is indeed a deciding pattern.

LEMMA 7.11. *If the Byzantine causal order prefix of a nonfaulty process p contains a deciding pattern such that each message in its first rank follows all of the messages in a candidate set S, then process p decides on S.*

Proof. Every message in the second rank of the deciding pattern follows exactly the same set of messages, aside from itself. Thus, if any message in the second rank or a previous message from its source votes on S at stage i, then every message in the second rank or a previous message from its source votes on S at stage i. Consequently, by Lemma 7.10, there exists a largest stage i at which every message in the second rank or a previous message from its source votes on S.

If any message in the second rank proposes on S at stage i, then all messages in the second rank propose on S at stage i and, furthermore, they all propose the same way. Four cases arise: (1) All messages in the second rank propose for S or all propose against S at stage i. This results in a decision at stage i based on the proposals by messages in the second rank. (2) All messages in the second rank propose indifferent to S at stage i. In this case, all messages in the third rank vote against S at stage $i+1$, and all messages in the fourth rank propose against S at stage $i+1$, resulting in a decision against S at stage $i+1$. (3) No message in the second rank proposes on S at stage i, and all messages in the third rank propose

for S or all propose against S at stage i, resulting in a decision at stage i. (4) No message in the second rank proposes on S at stage i and all messages in the third rank propose indifferent to S at stage i. In this case, all messages in the fourth rank vote against S at stage $i+1$, and all messages in the fifth rank propose against S at stage $i+1$, resulting in a decision against S at stage $i+1$. ▮

Lemma 7.12 shows that the probability that a Byzantine causal order prefix contains a deciding pattern that follows a candidate set increases as the size of the prefix increases, while Lemma 7.13 shows that the probability that a nonfaulty process selects a candidate set for the lth extension increases as the number of steps taken by the process increases.

LEMMA 7.12. *Let S be a candidate set, each message of which is followed by a message from a nonfaulty process q. The probability that a Byzantine causal order prefix of size x obtained by a nonfaulty process p contains a deciding pattern, such that each message in its first rank follows all of the messages in S, increases asymptotically to unity as x tends to infinity.*

Proof. The proof is identical to that of Lemma 5.12. ▮

LEMMA 7.13. *The probability that a nonfaulty process p selects a set S_p^l of candidate messages for the lth extension of its total order prefix T_p^l, contingent on its having constructed the $(l-1)$th extension of its total order prefix, increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Lemma 5.13. ▮

This leads directly to Theorems 7.3 and 7.4, which establish the Probabilistic Termination requirements.

THEOREM 7.3. *The probability that a nonfaulty process p places an ith message in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Theorem 5.3. ▮

THEOREM 7.4. *For each message m from a nonfaulty process q, the probability that a nonfaulty process p places message m in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Theorem 5.4. ▮

APPENDIX II

We present here the proof of correctness for the $2k_c + 3k_b$ algorithm given in Section 7. This proof is similar to that for the $3k_c + 3k_b$ algorithm.

LEMMA 8.1. *If message m is eligible to vote (propose) on S at stage i, then no mutant m' of m is eligible to vote (propose) on S at stage i.*

Proof. The proof for eligibility to vote is identical to that of Lemma 7.1, as is the proof for eligibility to propose. ▮

LEMMA 8.2. *If message m has no mutant, then m is eligible to vote (propose) on S at stage i.*

Proof. The proof for eligibility to vote is identical to that of Lemma 7.2, as is the proof for eligibility to propose. ▮

Received November 20, 1995; final manuscript received October 8, 1998

REFERENCES

[ADKM] Amir, Y., Dolev, D., Kramer, S., and Malki, D. (1992), Transis: A communication sub-system for high availability, *in* "Proceedings, 22nd Annual International Symposium on Fault-Tolerant Computing," pp. 76–84.

[BDGK] Bar-Noy, A., Deng, X., Garay, J. A., and Kameda, T. (1991), Optimal amortized distributed consensus, *Inform. and Comput.* **120**, 93–100.

[BD] Bar-Noy, A., and Dolev, D. (1991), Consensus algorithms with one-bit messages, *Distrib. Comput.* **4**, 105–110.

[BN] Bazzi, R., and Neiger, G. (1992), Simulating crash failures with many faulty processors, *in* "Proceedings, 6th International Workshop on Distributed Algorithms," Lecture Notes in Computer Science, Vol. 647, pp. 166–184, Springer-Verlag, Berlin/New York.

[B] Ben-Or, M. (1993), Another advantage of free choice: Completely asynchronous agreement protocols, *in* "Proceedings, 2nd Annual ACM Symposium on Principles of Distributed Computing," pp. 27–30.

[BV] Birman, K. P., and van Renesse, R. (1994), "Reliable Distributed Computing with the Isis Toolkit," IEEE Comput. Society Press, Los Alamitos, CA.

[B] Bracha, G. (1987), Asynchronous Byzantine agreement protocols, *Inform. and Comput.* **75**, 130–143.

[BT] Bracha, G., and Toueg, S. (1985), Asynchronous consensus and broadcast protocols, *J. Assoc. Comput. Mach.* **31**, 824–840.

[CT] Chandra, T. D., and Toueg, S. (1996), Unreliable failure detectors for asynchronous systems, *J. Assoc. Comput. Mach.* **43**, 225–267.

[CHT] Chandra, T. D., Hadzilacos, V., and Toueg, S. (1996), The weakest failure detector for solving consensus, *J. Assoc. Comput. Mach.* **43**, 685–722.

[CHD] Chokler, G. V., Huleihel, N., and Dolev, D. (1998), An adaptive totally ordered multicast protocol that tolerates partitions, *in* "Proceedings, 17th ACM Symposium on Principles of Distributed Computing," pp. 237–246.

[C] Cristian, F. (1991), Understanding fault-tolerant distributed systems, *Commun. Assoc. Comput. Mach.* **34**, 56–78.

[DKM] Dolev, D., Kramer, S., and Malki, D. (1993), Early delivery totally ordered multicast in asynchronous environments, *in* "Proceedings, 23rd Annual International Symposium on Fault-Tolerant Computing," pp. 544–553.

[FLP] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985), Impossibility of distributed consensus with one faulty process, *J. Assoc. Comput. Mach.* **32**, 374–382.

[GP] Garay, J. A., and Perry, K. J. (1992), A continuum of failure models for distributed computing, *in* "Proceedings, 6th International Workshop on Distributed Algorithms," Lecture Notes in Computer Science, Vol. 647, pp. 153–165, Springer-Verlag, Berlin/New York.

[GT] Gopal, A., and Toueg, S. (1989), Reliable broadcast in synchronous and asynchronous environments, *in* "Proceedings, 3rd International Workshop on Distributed Algorithms," Lecture Notes in Computer Science, Vol. 392, pp. 110–123, Springer-Verlag, Berlin/New York.

[KT] Kaashoek, M. F., and Tanenbaum, A. S. (1991), Group communication in the Amoeba distributed operating system, *in* "Proceedings, IEEE 11th International Conference on Distributed Computing Systems," pp. 882–891.

[KMM] Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (1998), The SecureRing protocols for securing group communication, *in* "Proceedings, Hawaii International Conference on System Sciences," Vol. 3, 317–326.

[L] Lamport, L. (1978), Time, clocks, and the ordering of events in a distributed system, *Commun. Assoc. Comput. Mach.* **21**, 558–565.

[MMA90] Melliar-Smith, P. M., Moser, L. E., and Agrawala, V. (1990), Broadcast protocols for distributed systems, *IEEE Trans. Parallel and Distributed Systems* **1**, 17–25.

[M] Meyer, F. J., and Pradhan, D. K. (1991), Consensus with dual failure modes, *IEEE Trans. Parallel Distrib. Systems* **2**, 214–222.

[MMA93] Moser, L. E., Melliar-Smith, P. M., and Agrawala, V. (1993), Asynchronous fault-tolerant total ordering algorithms, *SIAM J. Comput.* **22**, 727–750.

[MMA94] Moser, L. E., Melliar-Smith, P. M., and Agrawala, V. (1994), Processor membership in asynchronous distributed systems, *IEEE Trans. Parallel and Distributed Systems* **5**, 459–473.

[MMABL] Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., Budhia, R. K., and Lingley-Papadopoulis, C. A. (1996), The Totem system, *Communications Assoc. Comput. Mach.* **39**, 54–63.

[PSL] Pease, M., Shostak, R. E., and Lamport, L. (1980), Reaching agreement in the presence of faults, *J. Assoc. Comput. Mach.* **27**, 228–234.

[Ra] Rabin, M. O. (1983), Randomized Byzantine generals, *in* "Proceedings, 24th Annual Symposium on Foundations of Computer Science," pp. 403–409.

[Re] Reiter, M. K. (1995), The Rampart toolkit for building high-integrity services, "Theory and Practice in Distributed Systems," Lecture Notes in Computer Science, Vol. 938, pp. 99–110, Springer-Verlag, Berlin/New York.

[S] Schneider, F. B., Byzantine generals in action: Implementing fail-stop processors, *ACM Trans. Comput. Syst.* **2**, 145–154.

[TP] Thambidurai, P., and Park, Y. K. (1988), Interactive consistency with multiple failure modes, *in* "Proceedings, 7th IEEE Symposium on Reliable Distributed Systems," pp. 93–100.

[TPS] Toueg, S., Perry, K. J., and Srikanth, T. K. (1987), Fast distributed agreement, *SIAM J. Comput.* **16**, 445–457.

[VBM] van Renesse, R., Birman, K. P., and Maffeis, S. (1996), Horus: A flexible group communication system, *Communications Assoc. Comput. Mach.* **39**, 76–83.