Reprinted from INFORMATION AND CONTROL
All Rights Reserved by Academic Press, New York and London

Vol. 52, No. 3, March 1982
Printed in Belgium

An Efficient Algorithm for Byzantine Agreement without Authentication

DANNY DOLEV

IBM Research Laboratory, San Jose, California 95193

MICHAEL J. FISCHER*

Yale University, New Haven, Connecticut 06520

ROB FOWLER†

University of Washington, Seattle, Washington 98105

NANCY A. LYNCH**

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

AND

H. RAYMOND STRONG

IBM Research Laboratory, San Jose, California 95193

Byzantine Agreement involves a system of n processes, of which some t may be faulty. The problem is for the correct processes to agree on a binary value sent by a transmitter that may itself be one of the n processes. If the transmitter sends the same value to each process, then all correct processes must agree on that value, but in any case, they must agree on some value. An explicit solution not using authentication for $n = 3t + 1$ processes is given, using $2t + 3$ rounds and $O(t^3 \log t)$ message bits. This solution is easily extended to the general case of $n \geqslant 3t + 1$ to give a solution using $2t + 3$ rounds and $O(nt + t^3 \log t)$ message bits.

* This work was supported in part by the Office of Naval Research under Contract N00014-80-C-0221 through a subcontract from the University of Washington and by the National Science Foundation under Grant MCS81-16678.

† This work was supported in part by the National Science Foundation under Grant MCS80-04111.

** On leave from Georgia Institute of Technology, Atlanta, Georgia 30332.

0019-9958/82 $2.00

1. Introduction

In this paper we improve previous algorithms in Pease *et al.* (1980), Lamport *et al.* (1983), Dolev (1981, 1982), and Dolev and Strong (1982), for achieving agreement among multiple processes. The context for this agreement is a network of unreliable processes that have a means for conducting several synchronized rounds of information exchange, after which they must all agree on some set of information. We will assume for simplicity that this set of information consists of a single bit. The generalization to arbitrary messages is straightforward.

The type of agreement we will study is called Byzantine Agreement (Lamport *et al.*, 1983) unanimity (Dolev, 1981), or interactive consistency (Pease *et al.*, 1980). Byzantine Agreement is a kind of guaranteed multicast of *a* value in which one process sends its value (bit) to other participants and they exchange various messages in order to agree on exactly what value was sent. It results when all correctly operating processes are able to agree either on a value or on the conclusion that the originator of the value is faulty. It is called Byzantine because we make no assumption about the behavior of any undetected faulty processes. More explicitly, Byzantine Agreement is achieved when

(I) all correctly operating processes agree on the same value, and

(II) if the transmitter operates correctly, then all correctly operating processes agree on its value.

Implicit in (I) and (II) is the idea that the agreement is synchronous in the sense that all processes reach this agreement at the same time. In other words, there must be some real time at which each of the processes has completed the execution of its algorithm for reaching agreement, and this time must be known in advance.

Our algorithm will handle the worst case assumption that faulty processes are not predictable and possibly even malicious. Even if the correctly operating processes cannot identify the faulty processes, they must still reach Byzantine Agreement. The algorithm does not depend in any way on anticipated behavior of faulty processes.

Previous best algorithms for reaching Byzantine Agreement were presented in Dolev (1981, 1982). These algorithms are polynomial and reach agreement with and without using authentication, respectively. (Authentication here refers to any protocol that prevents processes from changing the messages they relay or introducing new messages and claiming to have received them from others. See Dolev (1982) and Lamport *et al.* (1982) for further discussion.) In this paper we will improve the algorithm without authentication, which requires $4t+4$ rounds of information exchange and

$O(n^4 \log n)$ bits. The basic algorithm we present requires $2t + 3$ rounds and $O(nt + t^3 \log t)$ bits.

Lynch and Fischer (1983) were the first to show that a lower bound on the number of rounds needed is $t + 1$; their result was generalized by Dolev and Strong (1982) and independently by Demillo *et al.* (1982). Exponential algorithms for reaching agreement in $t + 1$ rounds were given in Pease *et al.* (1983), Lamport *et al.* (1982), and Dolev (1981, 1982). The existence of a polynomial algorithm without authentication using fewer than $2t + 3$ rounds remains open.

The algorithms discussed provide a method for a single process to send a single value to all other processes. Generalizations to the cases in which the transmitter is not known in advance or is not a member of the system (one of the participating processes) will be discussed and the appropriate changes in the algorithm will be indicated.

We assume some reliable means of communication by which any correct process can send a message to any other correct process. For example, this reliablity might be achieved by sending duplicate messages along many paths in a network. In any case, for this paper we assume a completely connected, totally reliable communication network, and in counting the total number of messages sent, we ignore any duplication or repetition inherent in this communication medium. Note that we only count messages sent in accordance with the algorithm.

All results obtained in this paper can be extended to networks which are not complete using similar methods to those described in Dolev (1981, 1982). The number of rounds and likewise the number of messages will increase, but the algorithms will remain polynomial.

Algorithms for reaching Byzantine Agreement using authentication are relatively simple (Dolev and Strong, 1983) and require $O(nt)$ messages. For Byzantine Agreement without authentication, a much more sophisticated algorithm is needed. Byzantine Agreement is more difficult without authentication because faulty processes can change intermediate values, and because no process can identify with certainty all those that relayed a given message. *Without authentication*, it was shown in Pease *et al.* (1980) that n must be greater than $3t$ for Byzantine Agreement to be possible.

2. Basic Notions and Assumptions

To clarify the relationships among the processes, we use notions suggested in Dolev (1981). The transmitter sends its value to its receivers either directly or via others called relays. A process can be a transmitter, a receiver of a relay according to its function in the network with respect to a given message. A process is correct if it follows the specified algorithms; a correct transmitter is a correct process that sends the same value to all its receivers. A faulty process is a relay or a transmitter that is not correct.

We assume that each process knows the topology of the network and of each subnetwork on which we will run the algorithm. Weaker assumptions will require a more complex and less efficient algorithm along the lines discussed in Dolev (1982).

Although it is enough to assume that there exists an upper bound on the delay of relaying a message through a correct process, we also assume that the algorithm is synchronous in the sense that each process knows the beginning and ending times of each round and these times are synchronous throughout the system. For further discussion of this issue, see Lamport (1983) and Dolev (1982).

An important and apparently necessary assumption (Lamport, 1983) without authentication is that the immediate sender of any message can be identified.

There is no solution if the upper bound on the number of faults exceeds one third of the processes. To make the algorithm more efficient, we want it to run with respect to a given upper bound and not the maximal upper bound. Thus we assume that the upper bound t for the number of possible faults is a parameter of the algorithm that each process is given.

Observe that if the actual number of faulty processes is larger than the upper bound, then the algorithm may fail to reach Byzantine Agreement without alerting any correct process to that fact. This is not a problem with the specific algorithm we are going to present, but inherent in the Byzantine assumptions. There is no way, in general, to know if the transmitter had sent conflicting values, or whether all the evidence is coming out of faulty processes. The proofs of necessary conditions for the existence of a solution in Pease (1980), Lamport (1983), and Dolev (1981, 1982), are based on this fact.

In Section 3 we present the basic algorithm for the case $n = 3t + 1$ with the transmitter known. In Sections 4 and 5 we present a formal model and prove that the algorithm reaches Byzantine Agreement. In Section 6 we will generalize the algorithm to any $n > 3t$ and to the cases where the transmitter is not known in advance or not one of the processes. We also indicate the changes required when the number of possible values sent is a function of the number of processes.

3. The Algorithm

In this section we will present the basic algorithm for achieving Byzantine Agreement. In the next sections we will present the precise model and the complete proof of correctness of the algorithm. The algorithm presented here will handle the case $n = 3t + 1$, and will use the assumption that the transmitter is known and the set of possible values is $\{0, 1\}$. The value 0 will also be the default in case the agreement algorithm concludes that the transmitter is faulty.

At the beginning the transmitter sends a "*" message to all processes including itself to indicate that its value is 1. If its value is 0, it sends nothing during the first round. If the processes agree that "*" was sent, then 1 will be the final decision; otherwise, the final decision will be 0. Thus the algorithm is not symmetric in the two possible values.

In the algorithm we use two thresholds, LOW and HIGH, where $\text{LOW} = t + 1$ and $\text{HIGH} = 2t + 1$. If LOW processes support some assertion, then at least one correct process supports it. If HIGH processes support something, then at least LOW correct ones support it. These thresholds are used to govern the support offered by a correct process to assertions made by another process.

The basic idea in the algorithm is to prevent the faulty processes from introducing faulty values by asking for at least LOW $(= t + 1)$ confirmations of a fact before adding additional support and by asking for HIGH confirmations of a fact before assuming that there will be agreement on that fact. The LOW threshold will prevent a collusion of faulty processes from introducing spurious information without initial support from at least one correct process. To prevent faulty processes from introducing additional support to critical assertions too close to the end of the algorithm to be communicated to all correct processes, each correct process will require a proof of progress before it supports otherwise supportable assertions. Thus information released too late to correct processes will be ignored.

During the algorithm two types of messages wil be sent: a "*" message and messages consisting of the name of some process. The "*" represents the asertion that the transmitter has value 1, and a name represents the assertion that the named process has sent "*."

At the beginning of each round each process sends its messages to every process. Then it receives messages from the others and decides what to send at the next round. The notion of round as well as all the other notions we are using will be defined more precisely in the next section.

Each process keeps a record of all messages it has received. Consider this collection as held by one process p. Denote by W_x the set of processes that have sent to process p the message "x." We call W_x the set of *witnesses* to message x. Process j is a *direct supporter* for process r if j receives "*" directly from r. In this case, if j is correct, it will send the message "r" and p will include j in W_r. Process j is an *indirect supporter* for r if it has a set of witnesses to r of cardinality LOW, i.e., if $|W_r| \geqslant \text{LOW}$ for j.

Process p *confirms* r if the cardinality of the set of witnesses to "r" is at least HIGH. Each process p has a set (possibly empty) of confirmed processes which we denote by C.

The last notion we need is initiation, which means sending "*." A process p *initiates* at round k if either at round 1 it receives "*" from the transmitter, or by the end of round k the cardinality of the set of confirmed processes C, not including the transmitter, is at least LOW $+ \max(0, \lceil k/2 \rceil - 2)$.

We assume that whenever a process broadcasts a message to all others, it also sends one to itself, for purposes of recording its own messages. We also assume that correct processes follow the algorithm and send each message exactly once.

We now give the following rules for correct operation for each process:

(R1) At round 1 the transmitter s broadcasts its value v to all processes.

(R2) At any round $k > 1$, each process broadcasts the names of all processes for which it is either a direct or indirect supporter, and which it has not previously broadcast. If it initiates at the end of the previous round, it also sends the "*" message unless it has previously done so.

(R3) If a process confirms HIGH processes it *commits* to 1.

(R4) If, after round $2t + 3$, the value 1 is committed then agree on 1; otherwise, agree on 0.

The correctness of this algorithm is somewhat subtle and is proved in the following sections. For the proof we present a formal model in which we will define all the notions we use. Using the formal model we will prove the correctness of the algorithm. However, the following discussion should help to motivate the proof.

During the course of execution, processes from time to time initiate. This means that they know that the transmitter has sent "*" to some correct process and that they are proposing to commit (i.e., to accept). A process announces initiation by sending "*" to the other processes.

A process receiving "*" becomes a witness to the initiation of the sending process. A process can become an "indirect" witness to an initiation by hearing about it from at least LOW other processes, since then at least one of them must be correct. In either case, it broadcasts that fact to all processes, including itself. (The sending process will thus record itself as a witness at the same time that all other processes record it as a witness.)

On receiving message "r" from process j, process p records the fact that j claims to be a witness to the initiation of r. When at least HIGH processes have claimed to be witnesses to r, then r is confirmed. The confirming process then knows one of two things must be true: either r is correct and has indeed initiated, or r is faulty, but nevertheless has told at least LOW correct processes that it has initiated.

A process initiates at the beginning of the second round if it receives "*" from the transmitter during the first round. Thereafter, it can only initiate if it has confirmed sufficiently many initiations by other processes. This threshold number for initiation is LOW through round 4 and then increases by one every two rounds until it reaches HIGH-1. By that time, either LOW correct processes will have initiated or it is no longer possible for a correct process to initiate. In the former case, after three more rounds, every correct process will commit. In the latter case, no correct process can commit. The delicate part of the argument concerns these last two facts: initiating and committing are easy enough so that as soon as LOW correct processes initiate, then an avalanche begins which results in all correct processes initiating and committing a small number of rounds later. However, committing is hard enough so that no process commits in the last three rounds except as a result of an avalanche started earlier.

4. The Model

We model the method for reaching Byzantine Agreement on a synchronous system of automata. Such a system S is described by the following:

N the set of n processes;

Q_i the set of states for process i;

ι_i initial state for process i;

$v0_s, v1_s$ initial states for the transmitter s, indicating the transmitter initial state with values 0 or 1 respectively;

$q(r) \in Q_1 \times \cdots \times Q_n$ state vector of the n processes in round r;

$M_{i,j}$ the set of possible messages that process i might send to process j;

$\mu_{i,j}: Q_i \to M_{i,j}, i,j \in N^2$ the message generation functions;

$\delta_j: Q_j \times M_{1,j} \times \cdots \times M_{n,j} \to Q_j,\ \ j \in N$ the state transition functions.

Let T be a subset of N, where $|T| \geqslant n-t$, and let $v \in \{0, 1\}$ (T is the set of *correct* processes and v is the value of the transmitter).

A *round* of the computation takes place in two steps. First, each process i sends messages from $M_{i,j}$ to each process j. Second, each process changes state based on its old state and the messages it receives. Faulty processes can send arbitrary messages, so there are in general many possible computations that must satisfy the agreement and validity conditions below. A sequence of state vectors $q(1), q(2),..., q(d+1)$ is a *d-round(T, v)-computation* if there exists messages $m_{i,j}(r) \in M_{i,j}$, for $i,j \in N$ and $1 \leqslant r \leqslant d$, such that

(1) INITIALIZATION: If $v=0$, then $q_s(1)=v0_s$. If $v=1$, then $q_s(1)=v1_s$. Each $q_i(1)=\imath_i$, for $i \neq s$.

(2) CORRECT MESSAGES: For each r, $1 \leqslant r \leqslant d$, and each $i \in T$, $j \in N$, $m_{i,j}(r)=\mu_{i,j}(q_i(r))$.

(3) CORRECT TRANSITIONS: For each r, $1 \leqslant r \leqslant d$, and each $j \in T$, $q_j(r+1)=\delta_j(q_j(r), m_{1,j}(r),..., m_{N,j}(r))$.

We say that system S *reaches Byzantine Agreement in d rounds* if for every subset T of N, with $|T| \geqslant n-t$, every $v \in \{0, 1\}$, and every d-round (T, v)-computation $q(1),..., q(d+1)$, the final state vector $q(d+1)$ satisfies the following:

(1) AGREEMENT: If $i,j \in T$, then $q_i(d+1) \in F_i$ iff $q_j(d+1) \in F_j$.

(2) VALIDITY: If $s \in T$, then for all $i \in T$, $q_i(d+1) \in F_i$ iff $q_s(1)=v1_s$.

Accepting states F_i mean that process i will agree on 1. Any other state means agreement on 0. The validity requirement means that, if the transmitter is correct, then each correct process will agree on its value.

5. Basic Solution

Now assume $n=3t+1$. We describe a system S. Let $I=\{*\} \cup N$ be a set of *message items*. Messages are subsets of message items; thus we take $M_{i,j}=M=2^I$.

Each process remembers all the message items it has ever received from any of the processes. Formally, a *data entity* is a pair in $D=I \times N$ with first component a message item and second component the name of the process from which it was received. A *process state q* is a pair (data(q), round(q)), where data(q) is contained in D and round (q) is a positive integer. Thus, we take $Q_i=Q=2^D \times IN$. Thus, the process state is determined by the data recieved from the other processes, and the current round number. The initial states are $\imath_i=(\varnothing, 1)$, $v0_s=(\varnothing, 1)$, and $v1_s=(\{*, s)\}, 1)$. The transmitter's initial state with value 0 is not different from the initial state of the other processes. The initial state with value 1 will be used below to generate "*" messages from the transmitter to every process. The transition function is

$$\delta_i(q, m_1,..., m_n)=(\text{data}(q) \cup \{(x,j) \in D \mid x \in m_j\}, \text{round}(q)+1).$$

Thus the state change adds the new information received at the current round and increments the round number by one. Our assumption that the system is synchronous is used here to require that each process updates its state even when no information is received at some round.

The heart of the algorithm is the message generation function. First, define thresholds LOW $= t+1$ and HIGH $= 2t+1$. Let $q \in Q$, and let $x \in I$. We define

$$W_x(q) = \{j \in N \mid (x, j) \in \mathrm{data}(q)\},$$

the *witnesses* to x, and we let $w_x(q)|$. We define

$$C(q) = \{k \in N \mid w_k(q) \geqslant \mathrm{HIGH}, \text{ and } k \neq s\},$$

the *confirmed* processes, and we let $c(q) = |C(q)|$. Thus a process x, other than the transmitter, is confirmed if there are HIGH processes that have sent x. Notice that if x is confirmed for p, then every other process has at least LOW witnesses to x because at most t are faulty.

A process initiates if it supports the fact that the transmitter started with value 1. Let $f(x) = \max(0, \lceil x/2 \rceil - 2)$. Process i *initiates* in q if

(I1) $i \in W_*(q)$,

(I2) $c(q) \geqslant \mathrm{LOW} + f(\mathrm{round}(q))$, or

(I3) $s \in W_*(q)$ and round $(q) = 2$.

Thus a process initiates in state q if it initiated previously, it has enough support for the fact that the transmitter started with value 1 (without counting the transmitter itself), or it received "*" from the transmitter at the first round. (It may help to think of initiation as taking place at the end of the round that led to state q.)

Process i *commits* in q if

$$|\{k \in N \mid w_k(q) \geqslant \mathrm{HIGH}\}| \geqslant \mathrm{HIGH}.$$

This means that i has HIGH support for the fact that HIGH processes have sent "*."

Now we can define the message generation function and the accepting states. We define $\mu_{i,j}(q)$ to be the smalles set satisfying the following rules:

(M1) Initiation. If i initiates in q, then $* \in \mu_{i,j}(q)$.

(M2) Direct support. $W_*(q)$ is contained in $\mu_{i,j}(q)$.

(M3) Indirect support. If $w_k(q) \geqslant \mathrm{LOW}$, then $k \in \mu_{i,j}(q)$, $k \in N$.

So, the message generation function produces "*" if the process initiates. It produces the names of all the processes to which proces i is a witness. Processor i is either a direct supporter for j, meaning that it has directly received "*" from j; or it has reason to believe that j sent "*" (indirect support) in the form of LOW witnesses to the fact.

Note that since all messages are remembered, process i need not send process j the same message item twice. Thus the actual set of message items sent can be that generated by the message generation function minus those message items sent before.

Finally, $F_i = F = \{q \in Q \mid i \text{ commits in } q\}$. So, the accepting states are all the states in which a process confirms.

THEOREM 1. *System S reaches Byzantine Agreement in* $2t+3$ *rounds.*

The next section contains the proof of Theorem 1.

6. PROOF OF CORRECTNESS

Lemmas1–11 prove Theorem 1 and establish the correctness of the algorithm. All refer to a fixed (T, v)-computation $q(1),..., q(d+1)$, $d = 2t+3$, with associated messages $m_{i,j}(r)$, $i, j \in N$, $1 \leqslant r \leqslant d$. Lemma 1 is immediate from the definitions and is stated to focus attention on the monotonicity of sets W and C.

LEMMA 1. Let $1 \leqslant r' \leqslant r \leqslant d+1$, $i \in T$. *Then* $W_x(q_i(r'))$ *is contained in* $W_x(q_i(r))$ *for all* $x \in I$, *and* $C(q_i(r'))$ *is contained in* $C(q_i(r))$. *Thus if* $i \in T$ *initiates (commits) in* $q_i(r')$, *then* i *initiates (commits) in* $q_i(r)$.

In Lemma 2 we prove that within two rounds after a correct process initiates, it is confirmed by all correct processes.

LEMMA 2. *If* $i \in T - \{s\}$ *initiates in* $q_i(r)$, $1 \leqslant r \leqslant d-1$, *then* $i \in C(q_j(r+2))$ *for all* $j \in T$.

Proof. Let k be arbitrary process in T, then $i \in W_*(q_k(r+1))$ by rule M1). Similarly, $k \in W_i(q_j(r+2))$ by rule (M2), for all $j \in T$. Hence, $W_i(q_j(r+2))$ contains T. The lemma follows since $|T| \geqslant \text{HIGH}$. ∎

Lemma 3 proves that within two rounds after all the processes in T initiate, all T commit.

LEMMA 3. *Let* $0 \leqslant r \leqslant d-2$. *If all* $i \in T$ *initiate in* $q_i(r)$, *then all* $i \in T$ *commit in* $q_i(r+2)$.

Proof. Assume all $i \in T$ initiate in $q_i(r)$. By Lemma 2, $T - \{s\}$ is contained in $C(q_j(r+2))$ for all $j \in T$. If s is not in T, then $c(q_j(r+2)) \geqslant |T| \geqslant \text{HIGH}$ and we are done. If s is in T, then $w_s(q_j(r+2)) \geqslant \text{HIGH}$ and $c(q_j(r+2)) \geqslant \text{HIGH} - 1$. Thus even though only HIGH − 1 processes are confirmed, there is HIGH support for the fact that HIGH processes have sent "*" and each $j \in T$ commits. ∎

LEMMA 4. *Let $i, j, k \in T$, $x \in I$, and $1 \leqslant r \leqslant d+1$. Then $k \in W_x(q_i(r))$ iff $k \in W_x(q_j(r))$.*

The proof of Lemma 4 follows from an easy induction on r using the fact that correct processes always broadcast their messages to every process. Notice that at round 1 only the transmitter can possibly find itself as a witness.

Lemma 5 says that if one correct process has i confirmed at round r, then all correct processes wil have i confirmed by round $r+1$.

LEMMA 5. *Let $1 \leqslant r \leqslant d$. If $i \in C(q_k(r))$ for some $k \in T$, then $i \in C(q_j(r+1))$ for all $j \in T$.*

Proof. For every $k \in T$, $C(q_k(1)) = \varnothing$. Assume $r \geqslant 2$. Since $i \in C(q_k(r))$, there must be a set A contained in the intersection of T with $W_i(q_k(r))$ with $|A| = \mathrm{LOW}$. By Lemma 4, for every $j \in T$, A is contained in $W_i(q_j(r))$. Thus, $i \in m_{j,h}(r)$ for all $j, h \in T$ by rule (M3). Hence, $j \in W_i(q_h(r+1))$ for each $j, h \in T$, so $i \in C(q_h(r+1))$ for each $h \in T$. ▮

Next we prove that if the transmitter s is correct and initiates at round 1, then all correct processes commit at round 4.

LEMMA 6. *If $s \in T$ and $q_s(1) = v1_s$, then each $i \in T$ commits in $q_i(4)$.*

Proof. By (M2), $s \in W_*(q_j(2))$ for every $j \in T$. Therefore, by (I3), each $j \in T$ initiates in $q_j(2)$. By Lemma 3, each $j \in T$ commits in $q_j(4)$. ▮

Lemma 7 states that within 4 rounds after LOW correct processes initiate, all T commit.

LEMMA 7. *Let $0 < r \leqslant d-3$. If there is a set A contained in $T - \{s\}$ with $|A| = \mathrm{LOW}$, such that $i \in A$ initiates in $q_i(r)$, then each $j \in T$ commits in $q_j(r+4)$.*

Proof. Let r' be the least number such that all $i \in A$ initiate in $q_i(r')$. By Lemma 2, A is contained in $C(q_j(r'+2))$ for all $j \in T$. We now argue that every $j \in T$ initiates in $q_j(r'+2)$). It will then follow by Lemma 3 that j commits in $q_j(r'+4)$, and hence also in $q_j(r+4)$ by Lemma 1. If the transmitter is correct, then the desired conclusion holds for every r by Lemma 6, so assume the transmitter is faulty. At $r'=1$ no correct process initiates. If $r'=2$, then $c(q_j(r'+2)) \geqslant |A| = \mathrm{LOW} = \mathrm{LOW} + f(r'+2)$. If $r' > 2$, then there is some $k \in A$ such that k initiates in $q_k(r')$ using rule (I2), so $c(q_k(r')) \geqslant \mathrm{LOW} + f(r')$. Since r' is minimal, k is not in $C(q_k(r'))$. By Lemma 1 and 5, $C(q_j(r'+2))$ contains $C(q_k(r'))$ for all $j \in T$. By Lemma 2, $k \in C(q_j(r'+2))$. Hence, $c(q_j(r'+2)) \geqslant \mathrm{LOW} + f(r') + 1 = \mathrm{LOW} + f(r'+2)$. Thus j initiates in $q_j(r'+2)$ by rule (I2) as desired. ▮

We next note that no correct process can have LOW support for the fact that correct process i has initiated unless i has in fact initiated. The proof of Lemma 8 is straightforward induction on r.

LEMMA 8. *Let* $1 \leqslant r \leqslant d$, $i, j \in T$. *If i does not initiate in $q_i(r)$, then i is not in $W_*(q_j(r+1))$ and $w_i(q_j(r+2)) <$ LOW.*

Lemma 9 states that a correct process commits only after at least LOW correct processes initiate.

LEMMA 9. *Let* $r \geqslant 2$, $i \in T$, *and suppose i commits in $q_i(r)$. Then there is a set B contained in T with $|B| =$ LOW such that every $j \in B$ initiates in $q_j(r-1)$.*

Proof. $c(q_i(r)) \geqslant$ HIGH, so there is a set B contained in the intersection of T with $C(q_i(r))$ with $|B| =$ LOW. Each $j \in B$ has $w_j(q_i(r)) \geqslant$ HIGH; hence, by Lemmas 8 and 1, j initiates in $q_j(r-1)$. ∎

Lemmas 10 and 11 use the previous lemmas to complete the proof of Theorem 1. First we prove that by the end of the computation, if one correct process commits, then all commit. Later we prove that the system reaches Byzantine Agreement.

LEMMA 10. *If any $i \in T$ commits in $q_i(d+1)$, then all do.*

Proof. Assume $i \in T$ commits in $q_i(d+1)$. By Lemma 9, there is a set A contained in T with $|A| =$ LOW such that every $j \in A$ initiates in $q_j(d)$. The cases where $t = 0$ and the transmitter is correct are covered by Lemma 4. So assume that the transmitter is faulty and that $t > 0$, which implies that $d > 4$. Now consider the least r for which such a set A exists. If $r \leqslant d - 3$, we are done by Lemmas 7 and 1. Hence, suppose $r \geqslant d - 2 = 2t + 1$. We derive a contradiction, There must be $k \in A$ which initiates in $q_k(r)$ using rule (I2). Then $c(q_k(r)) \geqslant \text{LOW} + f(r) \geqslant \text{LOW} + t - 1 = \text{HIGH} - 1$; but the transmitter s is faulty and is not in $C(q_k(r))$. Therefore, as in the proof of Lemma 9, there is a set A' contained in the intersection of T with $C(q_k(r))$ with $|A'| =$ LOW such that every process $j \in A'$ initiates in $q_j(r-1)$, contradicting the choice of r. ∎

Lemma 10 proves the AGREEMENT part of the Byzantine Agreement. It remains to show that if the transmitter is correct, then all will reach an accepting state iff its value is 1.

LEMMA 11. *Assume that $s \in T$ and let $i \in T$.* (a) *If $q_s(1) = v0_s$, then $q_i(s+1)$ is not in F_i.* (b) *If $q_s(1) = v1_s$, then $q_i(d+1) \in F_i$.*

Proof. (a) $q_s(1) = v0_s$. Suppose i commits in $q_i(d+1)$. Then by

Lemma 9, there is an element $j \in T$ that initiates in $q_j(d)$. Consider the least r for which some $j \in T$ initiates in $q_j(r)$. Clearly $r > 1$ by the initial conditions. Moreover, j cannot be initiated by (I3). Hence, j initiates by rule (I2), so $c(q_j(r)) \geqslant$ LOW. Thus, there is a $k \in T$, a subset of $C(q_j(r))$, so $w_k(q_j) >$ HIGH. But then it follows from Lemma 8 that k initiates in $q_k(r-1)$, contradicting the choice of r. We conclude that $q_i(d+1)$ is not in F_i.

(b) $q_s(1) = v1_s$. This case is covered by Lemma 6. ▮

We have shown that the appropriate state for agreement is reached at $q(d+1)$ after $d = 2t+3$ rounds of information exchange. This completes the proof of Theorem 1. ▮

7. Complexity Analysis

Since $|I| = n+1$, each message item can be encoded by $O(\log n)$ bits, and a message M consisting of k message items can be encoded in length $O(k \log n)$. Since processes need not repeat messages, each process sends a maximum of $n+1$ message items to each other process during the course of the algorithm. Thus an upper bound on the total number of bits required by the algorithm is $O(n^2(n+1)\log n)) = O(t^3 \log t)$. We summarize this discussion in

Theorem 2. *Byzantine Agreement can be reached for* $3t+1$ *processors in the presence of up to* t *faults within* $2t+3$ *rounds of information exchange using* $O(t^3 \log t)$ *message bits for a one bit agreement.*

8. Generalizations

More Than $3t+1$ *Processes*

The first generalization will be to the case where the number of processes n is greater than $3t+1$. We can run the above algorithm with all the processes, but then the number of messages will be much larger than necessary. In order to reduce the total number of messages sent during the exchanges of information, we designate $3t+1$ *active* processes, including the transmitter. The other processes are called *passive*. Passive processes do not send messages. All processes are to ignore messages from or about passive processes. Active processes follow the basic algorithm and send "*" messages to all processes and other messages only to active processes. Passive processes agree on value 1 only if they receive "*" from HIGH ($=2t+1$) active processes; otherwise, they agree on value 0. After $2t+3$

rounds of information exchange, the active processes will have reached Byzantine Agreement by Theorem 1. If the transmitter is correct and has value 1, then all correct active processes initiate after round 1, so the correct passive processes will also agree on 1. If the transmitter is correct and has value 0, then no correct active process initiates by the proof of Lemma 11, so all correct passive processes will agree on 0. If the transmitter is faulty and some correct passive process receives "*" messages from HIGH distinct active processes, then at least LOW correct active processes initiated by the end of round $2t+2$. Following the proof of Lemma 10, it is easy to see that in this case at least LOW correct active processes initiated by the end of round $2t$, and by the proof of Lemma 7 every correct active process initiated by the end of round $2t+2$. Thus in this case, all correct processes will agree on 1. Similarly, if the transmitter is faulty and the correct active processes agree on 1, then so will the correct passive processes. In any case we have Byzantine Agreement for all n processes and this agreement is reached within $2t+3$ rounds of information exchange with tha number of message bits at most $O(nt + t^3 \log t)$.

Transmitter May Not Be a Process

We can assume about the transmitter only that it is a possibly faulty data source that communicates a (binary) value to each of the n processes in the system before the algorithm begins. Thus, the transmitter might be one of the n processes, or it might be a sensor or I/O device that all processes can read. In this formalization, the transmitter's value is encoded by the start state of each process. In the preceding sections the transmitter was identified with one of the n processes that carries out the algorithm, and each other process started in the same state regardless of the value of the transmitter. A solution for the version now under consideration can be modified to solve the previous version by simply adding an initial round in which the transmitter sends its value to each other process. The converse, however, is not in general true, for an algorithm might make use of the fact that, at most, $t-1$ faulty processes remain when the transmitter has been determined to be one of the faulty processes. Thus some solution to the previous version may not provide a solution to the current version.

However, our basic solution can be modified to handle this current version. If the transmitter is not one of the active processes, then the processes will not use its messages during the algorithm. The algorithm is the same, with the special phase at the beginning in which the data source loads values into the various processes. The number of active processes should be $3t+1$ not counting the transmitter if it is not one of the processes. At the crucial point in the proof of Lemma 10, we can no longer get by with HIGH $-$ 1 confirmed processes. Instead we need HIGH. Thus we need

$f(d-3)=t$ and $d=2t+6$, and the algorithm reaches Byzantine Agreement among the active processes after $2t+4$ rounds of information exchange (not counting the preloading round), with $m=O(t^3 \log t)$ bits of information exhanged.

Notice that the above variation also handles the case in which the transmitter is one of the processes, but is not identified ahead of time to the individual active processes.

More Than Two Possible Values for Agreement

Our last generalization is handling more than binary values. In this case each previous message must be relativized to the value v. Active processes can run the basic algorithm with respect to each value, and they will have accepting state for each value. At the end of the algorithm, if a process commits to exactly one value, this value will be its decision; in any other case it can decide that the transmitter is faulty and produce a previously agreed on default value. The handling of the previously considered generalizations remains the same.

Let V be the set of possible values. The number of bits required to reach Byzantine Agreement among the active processes is at most $O(|V|\, t^3(\log t + \log|V|))$. Note that if we use authentication, then the number of bits exchanged is not affected by the number of possible values (Dolev and Strong, 1938).

9. Decision on Transmitter Fault

In practice we would like to reach Byzantine Agreement as quickly as possible, but we would also like to identify faulty processes whenever possible. In this section we will sketch some methods for discovering transmitter fault as a byproduct of the algorithms we have presented.

Consider again the basic one bit case with a known transmitter and a set of $3t+1$ active processes. If one processor knows that the transmitter sent more than one value, then it knows that the transmitter is faulty. But to impart this knowledge to the other processors would seem to require something like another session of Byzantine Agreement. We will show how to use the information already exchanged to reach agreement among all the correct processes about faultiness of the transmitter in some cases.

Observe that not every transmitter fault can be detected. For example, if the transmitter excludes only t processes from receiving its value at the first round and follows the rest of the algorithm, then there is no way for an external observer to distinguish the resulting message behavior from what would happen if the transmitter were correct and the t processes faulty.

A correct transmitter sends the same value to every other process, and as we proved in Lemma 6, within 3 more rounds all correct processes commit to the value if it is 1. Committing to 0 can take place only at the end of the algorithm. A faulty transmitter might send "*" to only a few correct processes at round one, and by a slow propagation, the correct processes might commit to 1 only at the last phase. The improvement we suggest cannot change the number of rounds it will take to end the algorithm, because sometimes we need all the rounds to verify that no correct process will commit to something not yet known to the rest. But the propagation is slow only while fewer than $t+1$ correct processes have initiated. In this case at least $t+1$ correct processes did not receive "*" message. We will show below how correct processes will in this situation be able to commit to 0. If later they also commit to 1, then all correct processes will be able to agree that the transmitter is faulty.

Execute the basic algorithm for 1 and 0 at the same time, where the algorithm for 1 proceeds exactly as described in Section 5. Initiation messages will be "*1" and "*0." If a process does not initiate for 1 at round 2, then it initiates for 0. The rest of the algorithm remains the same.

In the algorithm for 0, each of the 11 lemmas hold, replacing 1 with 0. Therefore, all correct processes either agree on 0 or on1, and if a correct process sends its value all agree on its value. At the end of both algorithms, if a process commits in both to the same value, then it should agree on that value; otherwise, it should agree that the transmitter is faulty.

Observe that this variation of the algorithm requires twice the number of bits and the same number of phases. The interesting property of this algorithm is that if any process commits to a value in any one of the subalgorithms after round 5, then the final agreement will be that the transmitter is faulty.

THEOREM 3. *All correct processes commit to at least one of the two values* 0 *and* 1 *after round* 5. *Therefore, if any correct process commits to any value after round* 5, *all correct processes will decide that the transmitter is faulty.*

Proof. There are two cases possible:

(1) $t+1$ correct processes receive 1 at round 1, and

(2) $t+1$ correct processes do not receive 1 at round 1.

In either case at least $t+1$ correct processes will initiate at least one of the values at round 2. By Lemma 7, all correct processes will commit to that value after round 5. If any correct process commits to a value after round 5 it will then have committed to two values. By Theorem 1, all correct processes will commit to both values by the end of the algorithm. Thus the final decision will be "transmitter faulty." ▮

10. Conclusion

We have presented a feasible (polynomial) solution to the problem of reaching Byzantine Agreement without using authentication. Our basic solution handles a one bit agreement for n processes when the upper bound on the number of possible faults is t, with $n = 3t + 1$. We generalized this solution to handle many possible values for many more processes as long as n remains greater than $3t$. We also described ways to handle modifications to the model in which the transmitter is not one of the processes or is not known to the other processes in advance.

We believe that our solution characterizes the best known method for reaching Byzantine Agreement without authentication. However, we have not been able to establish a lower bound on the number of rounds required that narrows the gap between the known lower bound of $t + 1$ and the $2t + 3$ rounds required by our algorithm. In fact the lower bound of $t + 1$ is in some sense tight because algorithms exist for Byzantine Agreement without authentication that require only $t + 1$ rounds at the cost of requiring exponentially many messages. We leave open the question of a tradeoff between messages and rounds.

We have assumed complete and reliable communication among the processes. Note that this may be achieved in an unreliable and incompletely connected network (Dolev, 1982) and that if we are given an algorithm for reliable communication of a message using a number of messages polynomial in the number of processes, then we can convert that algorithm to one that achieves Byzantine Agreement in a polynomial number of messages.

We have also assumed a complete synchronization of the rounds throughout the paper. Some variations of our algorithm are more sensitive to changes in this assumption than others; but all can stand a significant weakening and the kind of synchronization required can be achieved by messages along the lines of Lamport (1983) assuming some known upper bound on the time required by a process to relay a message.

Finally, we offered an enhancement to the basic solution in which late activity in the algorithm allows all processes to conclude that the transmitter is faulty. Somewhat paradoxically, we can conclude from certain kinds of activity after round 6 that the final decision will be "transmitter fault," but we must continue processing to the end of the algorithm. This apparent paradox suggests as an area for further research the search for algorithms that under some conditions stop early.

Acknowledgments

We would like to acknowledge the careful reading and helpful suggestions of the referee.

References

DeMillo, Lynch, N. A., and Merritt, M. (1982), Cryptographic Protocols, *in* "Proc. 14th ACM SIGACT Symposium on Theory of Computing," May.

Dolev, D. (1982), The Byzantine Generals strike again, *J. Algorithms* **3**.

Dolev, D. (1981), Unanimity in an unknown and unreliable environment, *in* "Proc. 22nd Annual Symposium on Foundations of Computer Science," pp. 159–168.

Dolev, D, and Strong, H. R. (1982), Polynomial algorithms for multiple processor agreement, *in* "Proc., 14th ACM SIGACT Symposium on Theory of Computing," May.

Dolev, D., and Strong, H. R. (1983), Authenticated algorithms for Byzantine agreement, *Sci. Ind. Appl. Math. J. Comput.*, to appear.

Lamport, L. (1983), "Using Time Instead of Time-Out for Fault-Tolerant Distributed Systems," *ACM Trans. Programming Languages and Systems*, to appear.

Lamport, L., Shostak, R., and Pease, M. (1982), The Byzantine General's problem, *ACM Trans. Programming Languages and Systems* **4** (3), 382.

Lynch, N., and Fischer, M. (1983), "A Lower Bound for the Time to Assure Interactive Consistency," submitted.

Pease, M., Shostak, R., and Lamport, L. (1980), Reaching agreement in the presence of faults, *J. Assoc. Comput. Mach.* **27** (2), 228.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium