The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interactive consistency

1. INTRODUCTION

A reliable computer system must be able to cope with the failure of one or more of its components. A failed component may exhibit a type of behavior that is often overlooked—namely, sending conflicting information to different parts of the system. The problem of coping with this type of failure is expressed abstractly as the Byzantine Generals Problem. We devote the major part of the paper to a discussion of this abstract problem and conclude by indicating how our solutions can be used in implementing a reliable computer system.

We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals

This research was supported in part by the National Aeronautics and Space Administration under contract NAS1-15428 Mod. 3, the Ballistic Missile Defense Systems Command under contract DASG60-78-C-0046, and the Army Research Office under contract DAAG29-79-C-0102.
Authors' address: Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025.

may be traitors, trying to prevent the loyal generals from reaching agreement. The generals must have an algorithm to guarantee that

A. All loyal generals decide upon the same plan of action.

The loyal generals will all do what the algorithm says they should, but the traitors may do anything they wish. The algorithm must guarantee condition A regardless of what the traitors do.

The loyal generals should not only reach agreement, but should agree upon a reasonable plan. We therefore also want to insure that

B. A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Condition B is hard to formalize, since it requires saying precisely what a bad plan is, and we do not attempt to do so. Instead, we consider how the generals reach a decision. Each general observes the enemy and communicates his observations to the others. Let $v(i)$ be the information communicated by the ith general. Each general uses some method for combining the values $v(1), \ldots, v(n)$ into a single plan of action, where n is the number of generals. Condition A is achieved by having all generals use the same method for combining the information, and Condition B is achieved by using a robust method. For example, if the only decision to be made is whether to attack or retreat, then $v(i)$ can be General i's opinion of which option is best, and the final decision can be based upon a majority vote among them. A small number of traitors can affect the decision only if the loyal generals were almost equally divided between the two possibilities, in which case neither decision could be called bad.

While this approach may not be the only way to satisfy conditions A and B, it is the only one we know of. It assumes a method by which the generals communicate their values $v(i)$ to one another. The obvious method is for the ith general to send $v(i)$ by messenger to each other general. However, this does not work, because satisfying condition A requires that every loyal general obtain the same values $v(1), \ldots, v(n)$, and a traitorous general may send different values to different generals. For condition A to be satisfied, the following must be true:

1. Every loyal general must obtain the same information $v(1), \ldots, v(n)$.

Condition 1 implies that a general cannot necessarily use a value of $v(i)$ obtained directly from the ith general, since a traitorous ith general may send different values to different generals. This means that unless we are careful, in meeting condition 1 we might introduce the possibility that the generals use a value of $v(i)$ different from the one sent by the ith general—even though the ith general is loyal. We must not allow this to happen if condition B is to be met. For example, we cannot permit a few traitors to cause the loyal generals to base their decision upon the values "retreat", ..., "retreat" if every loyal general sent the value "attack". We therefore have the following requirement for each i:

2. If the ith general is loyal, then the value that he sends must be used by every loyal general as the value of $v(i)$.

We can rewrite condition 1 as the condition that for every i (whether or not the ith general is loyal),

1′. Any two loyal generals use the same value of $v(i)$.

Conditions 1′ and 2 are both conditions on the single value sent by the ith general. We can therefore restrict our consideration to the problem of how a single general sends his value to the others. We phrase this in terms of a commanding general sending an order to his lieutenants, obtaining the following problem.

Byzantine Generals Problem. A commanding general must send an order to his $n - 1$ lieutenant generals such that

IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Conditions IC1 and IC2 are called the *interactive consistency* conditions. Note that if the commander is loyal, then IC1 follows from IC2. However, the commander need not be loyal.

To solve our original problem, the ith general sends his value of $v(i)$ by using a solution to the Byzantine Generals Problem to send the order "use $v(i)$ as my value", with the other generals acting as the lieutenants.

2. IMPOSSIBILITY RESULTS

The Byzantine Generals Problem seems deceptively simple. Its difficulty is indicated by the surprising fact that if the generals can send only oral messages, then no solution will work unless more than two-thirds of the generals are loyal. In particular, with only three generals, no solution can work in the presence of a single traitor. An oral message is one whose contents are completely under the control of the sender, so a traitorous sender can transmit any possible message. Such a message corresponds to the type of message that computers normally send to one another. In Section 4 we consider signed, written messages, for which this is not true.

We now show that with oral messages no solution for three generals can handle a single traitor. For simplicity, we consider the case in which the only possible decisions are "attack" or "retreat". Let us first examine the scenario pictured in Figure 1 in which the commander is loyal and sends an "attack" order, but Lieutenant 2 is a traitor and reports to Lieutenant 1 that he received a "retreat" order. For IC2 to be satisfied, Lieutenant 1 must obey the order to attack.

Now consider another scenario, shown in Figure 2, in which the commander is a traitor and sends an "attack" order to Lieutenant 1 and a "retreat" order to Lieutenant 2. Lieutenant 1 does not know who the traitor is, and he cannot tell what message the commander actually sent to Lieutenant 2. Hence, the scenarios in these two pictures appear exactly the same to Lieutenant 1. If the traitor lies consistently, then there is no way for Lieutenant 1 to distinguish between these two situations, so he must obey the "attack" order in both of them. Hence, whenever Lieutenant 1 receives an "attack" order from the commander, he must obey it.

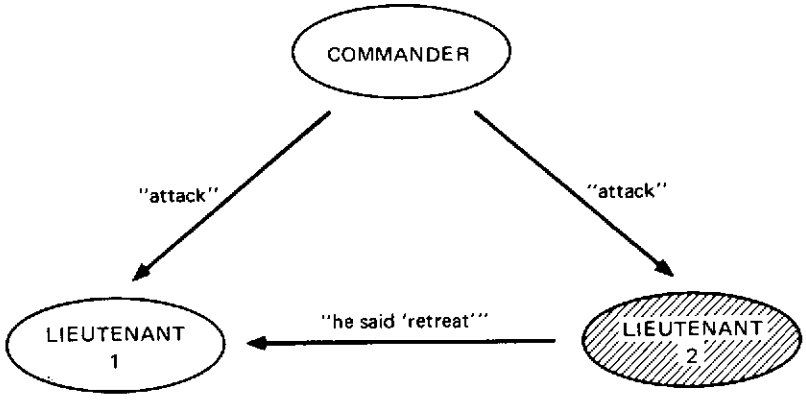

Fig. 1. Lieutenant 2 a traitor.

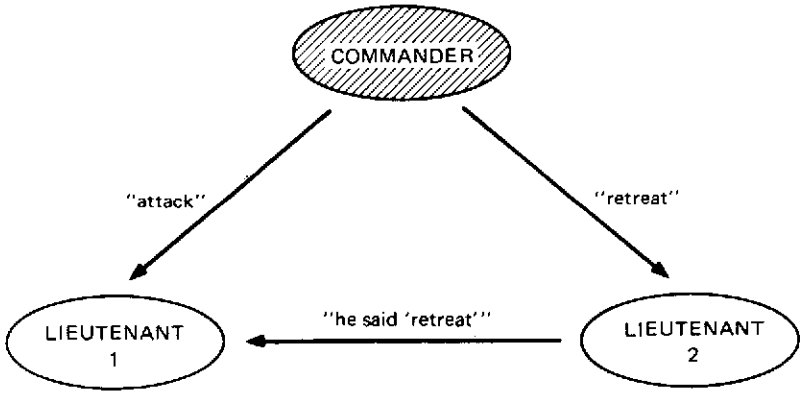

Fig. 2. The commander a traitor.

However, a similar argument shows that if Lieutenant 2 receives a "retreat" order from the commander then he must obey it even if Lieutenant 1 tells him that the commander said "attack". Therefore, in the scenario of Figure 2, Lieutenant 2 must obey the "retreat" order while Lieutenant 1 obeys the "attack" order, thereby violating condition IC1. Hence, no solution exists for three generals that works in the presence of a single traitor.

This argument may appear convincing, but we strongly advise the reader to be very suspicious of such nonrigorous reasoning. Although this result is indeed correct, we have seen equally plausible "proofs" of invalid results. We know of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm. For a rigorous proof of the impossibility of a three-general solution that can handle a single traitor, we refer the reader to [3].

Using this result, we can show that no solution with fewer than $3m + 1$ generals can cope with m traitors.[1] The proof is by contradiction—we assume such a

[1] More precisely, no such solution exists for three or more generals, since the problem is trivial for two generals.

solution for a group of $3m$ or fewer and use it to construct a three-general solution to the Byzantine Generals Problem that works with one traitor, which we know to be impossible. To avoid confusion between the two algorithms, we call the generals of the assumed solution Albanian generals, and those of the constructed solution Byzantine generals. Thus, starting from an algorithm that allows $3m$ or fewer Albanian generals to cope with m traitors, we construct a solution that allows three Byzantine generals to handle a single traitor.

The three-general solution is obtained by having each of the Byzantine generals simulate approximately one-third of the Albanian generals, so that each Byzantine general is simulating at most m Albanian generals. The Byzantine commander simulates the Albanian commander plus at most $m - 1$ Albanian lieutenants, and each of the two Byzantine lieutenants simulates at most m Albanian lieutenants. Since only one Byzantine general can be a traitor, and he simulates at most m Albanians, at most m of the Albanian generals are traitors. Hence, the assumed solution guarantees that IC1 and IC2 hold for the Albanian generals. By IC1, all the Albanian lieutenants being simulated by a loyal Byzantine lieutenant obey the same order, which is the order he is to obey. It is easy to check that conditions IC1 and IC2 of the Albanian generals solution imply the corresponding conditions for the Byzantine generals, so we have constructed the required impossible solution.

One might think that the difficulty in solving the Byzantine Generals Problem stems from the requirement of reaching exact agreement. We now demonstrate that this is not the case by showing that reaching approximate agreement is just as hard as reaching exact agreement. Let us assume that instead of trying to agree on a precise battle plan, the generals must agree only upon an approximate time of attack. More precisely, we assume that the commander orders the time of the attack, and we require the following two conditions to hold:

IC1′. All loyal lieutenants attack within 10 minutes of one another.
IC2′. If the commanding general is loyal, then every loyal lieutenant attacks within 10 minutes of the time given in the commander's order.

(We assume that the orders are given and processed the day before the attack and that the time at which an order is received is irrelevant—only the attack time given in the order matters.)

Like the Byzantine Generals Problem, this problem is unsolvable unless more than two-thirds of the generals are loyal. We prove this by first showing that if there were a solution for three generals that coped with one traitor, then we could construct a three-general solution to the Byzantine Generals Problem that also worked in the presence of one traitor. Suppose the commander wishes to send an "attack" or "retreat" order. He orders an attack by sending an attack time of 1:00 and orders a retreat by sending an attack time of 2:00, using the assumed algorithm. Each lieutenant uses the following procedure to obtain his order.

(1) After receiving the attack time from the commander, a lieutenant does one of the following:

   (a) If the time is 1:10 or earlier, then attack.
   (b) If the time is 1:50 or later, then retreat.
   (c) Otherwise, continue to step (2).

(2) Ask the other lieutenant what decision he reached in step (1).

(a) If the other lieutenant reached a decision, then make the same decision he did.
(b) Otherwise, retreat.

It follows from IC2′ that if the commander is loyal, then a loyal lieutenant will obtain the correct order in step (1), so IC2 is satisfied. If the commander is loyal, then IC1 follows from IC2, so we need only prove IC1 under the assumption that the commander is a traitor. Since there is at most one traitor, this means that both lieutenants are loyal. It follows from IC1′ that if one lieutenant decides to attack in step (1), then the other cannot decide to retreat in step (1). Hence, either they will both come to the same decision in step (1) or at least one of them will defer his decision until step (2). In this case, it is easy to see that they both arrive at the same decision, so IC1 is satisfied. We have therefore constructed a three-general solution to the Byzantine Generals Problem that handles one traitor, which is impossible. Hence, we cannot have a three-general algorithm that maintains IC1′ and IC2′ in the presence of a traitor.

The method of having one general simulate m others can now be used to prove that no solution with fewer than $3m + 1$ generals can cope with m traitors. The proof is similar to the one for the original Byzantine Generals Problem and is left to the reader.

3. A SOLUTION WITH ORAL MESSAGES

We showed above that for a solution to the Byzantine Generals Problem using oral messages to cope with m traitors, there must be at least $3m + 1$ generals. We now give a solution that works for $3m + 1$ or more generals. However, we first specify exactly what we mean by "oral messages". Each general is supposed to execute some algorithm that involves sending messages to the other generals, and we assume that a loyal general correctly executes his algorithm. The definition of an oral message is embodied in the following assumptions which we make for the generals' message system:

A1. Every message that is sent is delivered correctly.
A2. The receiver of a message knows who sent it.
A3. The absence of a message can be detected.

Assumptions A1 and A2 prevent a traitor from interfering with the communication between two other generals, since by A1 he cannot interfere with the messages they do send, and by A2 he cannot confuse their intercourse by introducing spurious messages. Assumption A3 will foil a traitor who tries to prevent a decision by simply not sending messages. The practical implementation of these assumptions is discussed in Section 6.

The algorithms in this section and in the following one require that each general be able to send messages directly to every other general. In Section 5, we describe algorithms which do not have this requirement.

A traitorous commander may decide not to send any order. Since the lieutenants must obey some order, they need some default order to obey in this case. We let RETREAT be this default order.

We inductively define the *O*ral *M*essage algorithms OM(m), for all nonnegative integers m, by which a commander sends an order to $n - 1$ lieutenants. We show that OM(m) solves the Byzantine Generals Problem for $3m + 1$ or more generals in the presence of at most m traitors. We find it more convenient to describe this algorithm in terms of the lieutenants "obtaining a value" rather than "obeying an order".

The algorithm assumes a function *majority* with the property that if a majority of the values v_i equal v, then *majority*$(v_1, \ldots, v_{n-1})$ equals v. (Actually, it assumes a sequence of such functions—one for each n.) There are two natural choices for the value of *majority*$(v_1, \ldots, v_{n-1})$:

1. The majority value among the v_i if it exists, otherwise the value RETREAT;
2. The median of the v_i, assuming that they come from an ordered set.

The following algorithm requires only the aforementioned property of *majority*.

Algorithm OM(0).

(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm OM(m), $m > 0$.

(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM($m - 1$) to send the value v_i to each of the $n - 2$ other lieutenants.
(3) For each i, and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM($m - 1$)), or else RETREAT if he received no such value. Lieutenant i uses the value *majority*$(v_1, \ldots, v_{n-1})$.

To understand how this algorithm works, we consider the case $m = 1$, $n = 4$. Figure 3 illustrates the messages received by Lieutenant 2 when the commander sends the value v and Lieutenant 3 is a traitor. In the first step of OM(1), the commander sends v to all three lieutenants. In the second step, Lieutenant 1 sends the value v to Lieutenant 2, using the trivial algorithm OM(0). Also in the second step, the traitorous Lieutenant 3 sends Lieutenant 2 some other value x. In step 3, Lieutenant 2 then has $v_1 = v_2 = v$ and $v_3 = x$, so he obtains the correct value $v =$ *majority*(v, v, x).

Next, we see what happens if the commander is a traitor. Figure 4 shows the values received by the lieutenants if a traitorous commander sends three arbitrary values x, y, and z to the three lieutenants. Each lieutenant obtains $v_1 = x$, $v_2 = y$, and $v_3 = z$, so they all obtain the same value *majority*(x, y, z) in step (3), regardless of whether or not any of the three values x, y, and z are equal.

The recursive algorithm OM(m) invokes $n - 1$ separate executions of the algorithm OM($m - 1$), each of which invokes $n - 2$ executions of OM($m - 2$), etc. This means that, for $m > 1$, a lieutenant sends many separate messages to each other lieutenant. There must be some way to distinguish among these different messages. The reader can verify that all ambiguity is removed if each lieutenant i prefixes the number i to the value v_i that he sends in step (2). As the recursion "unfolds," the algorithm OM($m - k$) will be called $(n - 1) \cdots (n - k)$ times to send a value prefixed by a sequence of k lieutenants' numbers.

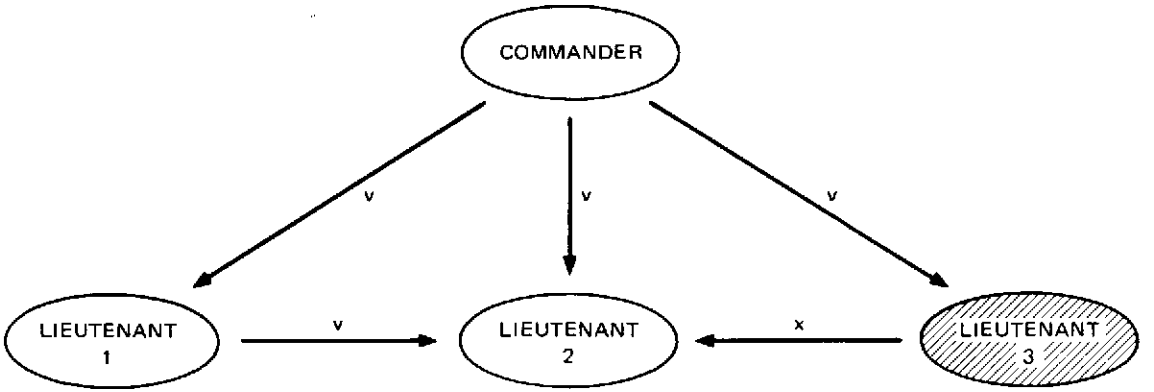

Fig. 3. Algorithm OM(1); Lieutenant 3 a traitor.

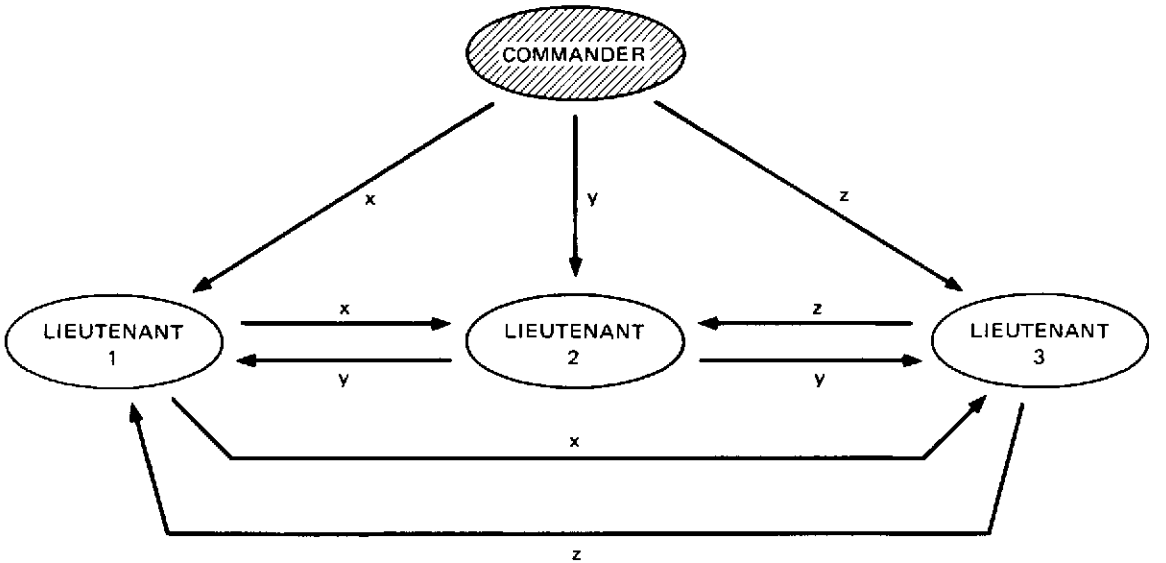

Fig. 4. Algorithm OM(1); the commander a traitor.

To prove the correctness of the algorithm OM(m) for arbitrary m, we first prove the following lemma.

LEMMA 1. *For any m and k, Algorithm OM(m) satisfies IC2 if there are more than $2k + m$ generals and at most k traitors.*

PROOF. The proof is by induction on m. IC2 only specifies what must happen if the commander is loyal. Using A1, it is easy to see that the trivial algorithm OM(0) works if the commander is loyal, so the lemma is true for $m = 0$. We now assume it is true for $m - 1$, $m > 0$, and prove it for m.

In step (1), the loyal commander sends a value v to all $n - 1$ lieutenants. In step (2), each loyal lieutenant applies OM($m - 1$) with $n - 1$ generals. Since by hypothesis $n > 2k + m$, we have $n - 1 > 2k + (m - 1)$, so we can apply the induction hypothesis to conclude that every loyal lieutenant gets $v_j = v$ for each loyal Lieutenant j. Since there are at most k traitors, and $n - 1 > 2k + (m - 1)$ $\geq 2k$, a majority of the $n - 1$ lieutenants are loyal. Hence, each loyal lieutenant has $v_i = v$ for a majority of the $n - 1$ values i, so he obtains *majority*$(v_1, \ldots, v_{n-1}) = v$ in step (3), proving IC2. □

The following theorem asserts that Algorithm OM(m) solves the Byzantine Generals Problem.

THEOREM 1. *For any m, Algorithm OM(m) satisfies conditions IC1 and IC2 if there are more than $3m$ generals and at most m traitors.*

PROOF. The proof is by induction on m. If there are no traitors, then it is easy to see that OM(0) satisfies IC1 and IC2. We therefore assume that the theorem is true for OM($m - 1$) and prove it for OM(m), $m > 0$.

We first consider the case in which the commander is loyal. By taking k equal to m in Lemma 1, we see that OM(m) satisfies IC2. IC1 follows from IC2 if the commander is loyal, so we need only verify IC1 in the case that the commander is a traitor.

There are at most m traitors, and the commander is one of them, so at most $m - 1$ of the lieutenants are traitors. Since there are more than $3m$ generals, there are more than $3m - 1$ lieutenants, and $3m - 1 > 3(m - 1)$. We may therefore apply the induction hypothesis to conclude that OM($m - 1$) satisfies conditions IC1 and IC2. Hence, for each j, any two loyal lieutenants get the same value for v_j in step (3). (This follows from IC2 if one of the two lieutenants is Lieutenant j, and from IC1 otherwise.) Hence, any two loyal lieutenants get the same vector of values $v_1, \ldots, v_{n-1}$, and therefore obtain the same value *majority*$(v_1, \ldots, v_{n-1})$ in step (3), proving IC1. □

4. A SOLUTION WITH SIGNED MESSAGES

As we saw from the scenario of Figures 1 and 2, it is the traitors' ability to lie that makes the Byzantine Generals Problem so difficult. The problem becomes easier to solve if we can restrict that ability. One way to do this is to allow the generals to send unforgeable signed messages. More precisely, we add to A1–A3 the

following assumption:

A4 (a) A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.
(b) Anyone can verify the authenticity of a general's signature.

Note that we make no assumptions about a traitorous general's signature. In particular, we allow his signature to be forged by another traitor, thereby permitting collusion among the traitors.

Now that we have introduced signed messages, our previous argument that four generals are required to cope with one traitor no longer holds. In fact, a three-general solution does exist. We now give an algorithm that copes with m traitors for any number of generals. (The problem is vacuous if there are fewer than $m + 2$ generals.)

In our algorithm, the commander sends a signed order to each of his lieutenants. Each lieutenant then adds his signature to that order and sends it to the other lieutenants, who add their signatures and send it to others, and so on. This means that a lieutenant must effectively receive one signed message, make several copies of it, and sign and send those copies. It does not matter how these copies are obtained; a single message might be photocopied, or else each message might consist of a stack of identical messages which are signed and distributed as required.

Our algorithm assumes a function *choice* which is applied to a set of orders to obtain a single one. The only requirements we make for this function are

1. If the set V consists of the single element v, then $choice(V) = v$.
2. $choice(\varnothing)$ = RETREAT, where $\varnothing$ is the empty set.

Note that one possible definition is to let $choice(V)$ be the median element of V—assuming that there is an ordering of the elements.

In the following algorithm, we let $x:i$ denote the value x signed by General i. Thus, $v:j:i$ denotes the value v signed by j, and then that value $v:j$ signed by i. We let General 0 be the commander. In this algorithm, each lieutenant i maintains a set V_i, containng the set of properly signed orders he has received so far. (If the commander is loyal, then this set should never contain more than a single element.) Do not confuse V_i, the set of *orders* he has received, with the set of messages that he has received. There may be many different messages with the same order.

Algorithm SM(m).

Initially $V_i = \varnothing$.

(1) The commander signs and sends his value to every lieutenant.
(2) For each i:

(A) If Lieutenant i receives a message of the form $v:0$ from the commander and he has not yet received any order, then

(i) he lets V_i equal $\{v\}$;
(ii) he sends the message $v:0:i$ to every other lieutenant.

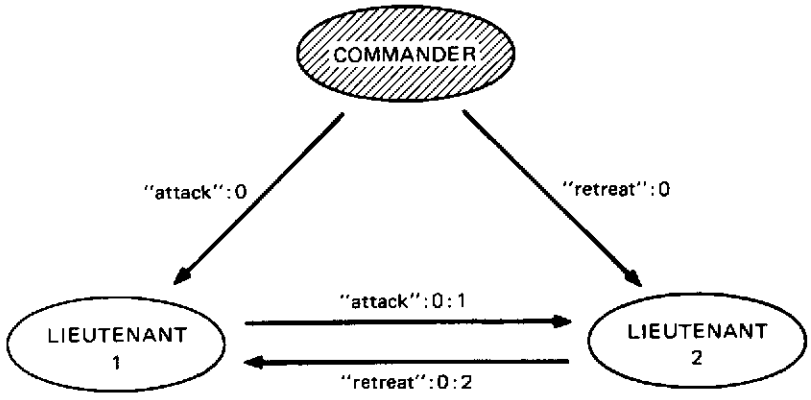

Fig. 5. Algorithm SM(1); the commander a traitor.

(B) If Lieutenant i receives a message of the form $v:0:j_1:\cdots:j_k$ and v is not in the set V_i, then
  (i) he adds v to V_i;
  (ii) if $k < m$, then he sends the message $v:0:j_1:\cdots:j_k:i$ to every lieutenant other than $j_1, \ldots, j_k$.

(3) For each i: When Lieutenant i will receive no more messages, he obeys the order $choice(V_i)$.

Note that in step (2), Lieutenant i ignores any message containing an order v that is already in the set V_i.

We have not specified how a lieutenant determines in step (3) that he will receive no more messages. By induction on k, one easily shows that for each sequence of lieutenants $j_1, \ldots, j_k$ with $k \leq m$, a lieutenant can receive at most one message of the form $v:0:j_1:\cdots:j_k$ in step (2). If we require that Lieutenant j_k either send such a message or else send a message reporting that he will not send such a message, then it is easy to decide when all messages have been received. (By assumption A3, a lieutenant can determine if a traitorous lieutenant j_k sends neither of those two messages.) Alternatively, time-out can be used to determine when no more messages will arrive. The use of time-out is discussed in Section 6.

Note that in step (2), Lieutenant i ignores any messages that do not have the proper form of a value followed by a string of signatures. If packets of identical messages are used to avoid having to copy messages, this means that he throws away any packet that does not consist of a sufficient number of identical, properly signed messages. (There should be $(n-k-2)(n-k-3)\cdots(n-m-2)$ copies of the message if it has been signed by k lieutenants.)

Figure 5 illustrates Algorithm SM(1) for the case of three generals when the commander is a traitor. The commander sends an "attack" order to one lieutenant and a "retreat" order to the other. Both lieutenants receive the two orders in step (2), so after step (2) $V_1 = V_2 = \{$"attack", "retreat"$\}$, and they both obey the order $choice(\{$"attack", "retreat"$\})$. Observe that here, unlike the situation in Figure 2, the lieutenants know the commander is a traitor because his signature

appears on two different orders, and A4 states that only he could have generated those signatures.

In Algorithm SM(m), a lieutenant signs his name to acknowledge his receipt of an order. If he is the mth lieutenant to add his signature to the order, then that signature is not relayed to anyone else by its recipient, so it is superfluous. (More precisely, assumption A2 makes it unnecessary.) In particular, the lieutenants need not sign their messages in SM(1).

We now prove the correctness of our algorithm.

THEOREM 2. *For any m, Algorithm SM(m) solves the Byzantine Generals Problem if there are at most m traitors.*

PROOF. We first prove IC2. If the commander is loyal, then he sends his signed order $v:0$ to every lieutenant in step (1). Every loyal lieutenant will therefore receive the order v in step (2)(A). Moreover, since no traitorous lieutenant can forge any other message of the form $v':0$, a loyal lieutenant can receive no additional order in step (2)(B). Hence, for each loyal Lieutenant i, the set V_i obtained in step (2) consists of the single order v, which he will obey in step (3) by property 1 of the *choice* function. This proves IC2.

Since IC1 follows from IC2 if the commander is loyal, to prove IC1 we need only consider the case in which the commander is a traitor. Two loyal lieutenants i and j obey the same order in step (3) if the sets of orders V_i and V_j that they receive in step (2) are the same. Therefore, to prove IC1 it suffices to prove that, if i puts an order v into V_i in step (2), then j must put the same order v into V_j in step (2). To do this, we must show that j receives a properly signed message containing that order. If i receives the order v in step (2)(A), then he sends it to j in step (2)(A)(ii); so j receives it (by A1). If i adds the order to V_i in step (2)(B), then he must receive a first message of the form $v:0:j_1:\cdots:j_k$. If j is one of the j_r, then by A4 he must already have received the order v. If not, we consider two cases:

1. $k < m$. In this case, i sends the message $v:0:j_1:\cdots:j_k:i$ to j; so j must receive the order v.

2. $k = m$. Since the commander is a traitor, at most $m - 1$ of the lieutenants are traitors. Hence, at least one of the lieutenants $j_1, \ldots, j_m$ is loyal. This loyal lieutenant must have sent j the value v when he first received it, so j must therefore receive that value.

This completes the proof. □

5. MISSING COMMUNICATION PATHS

Thus far, we have assumed that a general can send messages directly to every other general. We now remove this assumption. Instead, we suppose that physical barriers place some restrictions on who can send messages to whom. We consider the generals to form the nodes of a simple,[2] finite undirected graph G, where an arc between two nodes indicates that those two generals can send messages

[2] A simple graph is one in which there is at most one arc joining any two nodes, and every arc connects two distinct nodes.

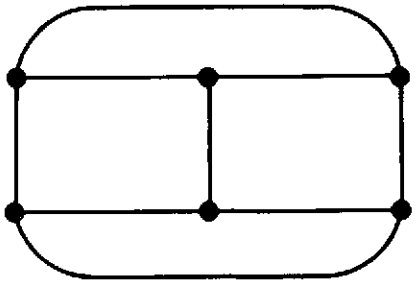

Fig. 6. A 3-regular graph.

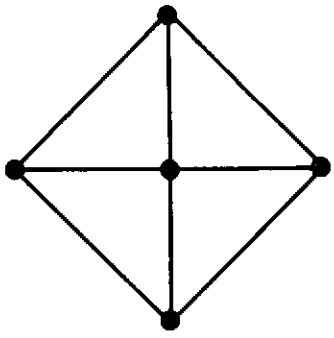

Fig. 7. A graph that is not 3-regular.

directly to one another. We now extend Algorithms OM(m) and SM(m), which assumed G to be completely connected, to more general graphs.

To extend our oral message algorithm OM(m), we need the following definition, where two generals are said to be *neighbors* if they are joined by an arc.

Definition 1.

(a) A set of nodes $\{i_1, \ldots, i_p\}$ is said to be a *regular set of neighbors* of a node i if

(i) each i_j is a neighbor of i, and
(ii) for any general k different from i, there exist paths $\gamma_{j,k}$ from i_j to k not passing through i such that any two different paths $\gamma_{j,k}$ have no node in common other than k.

(b) The graph G is said to be *p-regular* if every node has a regular set of neighbors consisting of p distinct nodes.

Figure 6 shows an example of a simple 3-regular graph. Figure 7 shows an example of a graph that is not 3-regular because the central node has no regular set of neighbors containing three nodes.

We extend OM(m) to an algorithm that solves the Byzantine Generals Problem in the presence of m traitors if the graph G of generals is $3m$-regular. (Note that a $3m$-regular graph must contain at least $3m + 1$ nodes.) For all *positive* integers m and p, we define the algorithm OM(m, p) as follows when the graph G of generals is p-regular. (OM(m, p) is not defined if G is not p-regular.) The definition uses induction on m.

Algorithm *OM*(m, p).

(0) Choose a regular set N of neighbors of the commander consisting of p lieutenants.
(1) The commander sends his value to every lieutenant in N.
(2) For each i in N, let v_i be the value Lieutenant i receives from the commander, or else RETREAT if he receives no value. Lieutenant i sends v_i to every other lieutenant k as follows:
    (A) If $m = 1$, then by sending the value along the path $\gamma_{i,k}$ whose existence is guaranteed by part (a)(ii) of Definition 1.
    (B) If $m > 1$, then by acting as the commander in the algorithm OM($m - 1$, $p - 1$), with the graph of generals obtained by removing the original commander from G.

(3) For each k, and each i in N with $i \neq k$, let v_i be the value Lieutenant k received from Lieutenant i in step (2), or RETREAT if he received no value. Lieutenant k uses the value $majority(v_{i_1}, \ldots, v_{i_p})$, where $N = \{i_1, \ldots, i_p\}$.

Note that removing a single node from a p-regular graph leaves a $(p - 1)$-regular graph. Hence, one can apply the algorithm OM$(m - 1, p - 1)$ in step (2)(B).

We now prove that OM$(m, 3m)$ solves the Byzantine Generals Problem if there are at most m traitors. The proof is similar to the proof for the algorithm OM(m) and will just be sketched. It begins with the following extension of Lemma 1.

LEMMA 2. *For any $m > 0$ and any $p \geq 2k + m$, Algorithm OM(m, p) satisfies IC2 if there are at most k traitors.*

PROOF. For $m = 1$, observe that a lieutenant obtains the value $majority(v_1, \ldots, v_p)$, where each v_i is a value sent to him by the commander along a path disjoint from the path used to send the other values to him. Since there are at most k traitors and $p \geq 2k + 1$, more than half of those paths are composed entirely of loyal lieutenants. Hence, if the commander is loyal, then a majority of the values v_i will equal the value he sent, which implies that IC2 is satisfied.

Now assume the lemma for $m - 1$, $m > 1$. If the commander is loyal, then each of the p lieutenants in N gets the correct value. Since $p > 2k$, a majority of them are loyal, and by the induction hypothesis each of them sends the correct value to every loyal lieutenant. Hence, each loyal lieutenant gets a majority of correct values, thereby obtaining the correct value in step (3). □

The correctness of Algorithm OM$(m, 3m)$ is an immediate consequence of the following result.

THEOREM 3. *For any $m > 0$ and any $p \geq 3m$, Algorithm OM(m, p) solves the Byzantine Generals Problem if there are at most m traitors.*

PROOF. By Lemma 2, letting $k = m$, we see that OM(m, p) satisfies IC2. If the commander is loyal, then IC1 follows from IC2, so we need only prove IC1 under the assumption that the commander is a traitor. To do this, we prove that every loyal lieutenant gets the same set of values v_i in step (3). If $m = 1$, then this follows because all the lieutenants, including those in N, are loyal and the paths $\gamma_{i,k}$ do not pass through the commander. For $m > 1$, a simple induction argument can be applied, since $p \geq 3m$ implies that $p - 1 \geq 3(m - 1)$. □

Our extension of Algorithm OM(m) requires that the graph G be $3m$-regular, which is a rather strong connectivity hypothesis.[3] In fact, if there are only $3m + 1$ generals (the minimum number required), then $3m$-regularity means complete connectivity, and Algorithm OM$(m, 3m)$ reduces to Algorithm OM(m). In contrast, Algorithm SM(m) is easily extended to allow the weakest possible connectivity hypothesis. Let us first consider how much connectivity is needed for the Byzantine Generals Problem to be solvable. IC2 requires that a loyal lieutenant obey a loyal commander. This is clearly impossible if the commander cannot communicate with the lieutenant. In particular, if every message from the

[3] A recent algorithm of Dolev [2] requires less connectivity.

commander to the lieutenant must be relayed by traitors, then there is no way to guarantee that the lieutenant gets the commander's order. Similarly, IC1 cannot be guaranteed if there are two lieutenants who can only communicate with one another via traitorous intermediaries.

The weakest connectivity hypothesis for which the Byzantine Generals Problem is solvable is that the subgraph formed by the loyal generals be connected. We show that under this hypothesis, the algorithm SM($n - 2$) is a solution, where n is the number of generals—regardless of the number of traitors. Of course, we must modify the algorithm so that generals only send messages to where they can be sent. More precisely, in step (1), the commander sends his signed order only to his neighboring lieutenants; and in step (2)(B), Lieutenant i only sends the message to every *neighboring* lieutenant not among the j_r.

We prove the following more general result, where the *diameter* of a graph is the smallest number d such that any two nodes are connected by a path containing at most d arcs.

THEOREM 4. *For any m and d, if there are at most m traitors and the subgraph of loyal generals has diameter d, then Algorithm SM($m + d - 1$) (with the above modification) solves the Byzantine Generals Problem.*

PROOF. The proof is quite similar to that of Theorem 2 and is just sketched here. To prove IC2, observe that by hypothesis there is a path from the loyal commander to a lieutenant i going through $d - 1$ or fewer loyal lieutenants. Those lieutenants will correctly relay the order until it reaches i. As before, assumption A4 prevents a traitor from forging a different order.

To prove IC1, we assume the commander is a traitor and must show that any order received by a loyal lieutenant i is also received by a loyal lieutenant j. Suppose i receives an order $v:0:j_1:\cdots:j_k$ not signed by j. If $k < m$, then i will send it to every neighbor who has not already received that order, and it will be relayed to j within $d - 1$ more steps. If $k \geq m$, then one of the first m signers must be loyal and must have sent it to all of his neighbors, whereupon it will be relayed by loyal generals and will reach j within $d - 1$ steps. □

COROLLARY. *If the graph of loyal generals is connected, then SM($n - 2$) (as modified above) solves the Byzantine Generals Problem for n generals.*

PROOF. Let d be the diameter of the graph of loyal generals. Since the diameter of a connected graph is less than the number of nodes, there must be more than d loyal generals and fewer than $n - d$ traitors. The result follows from the theorem by letting $m = n - d - 1$. □

Theorem 4 assumes that the subgraph of loyal generals is connected. Its proof is easily extended to show that even if this is not the case, if there are at most m traitors, then the algorithm SM($m + d - 1$) has the following two properties:

1. Any two loyal generals connected by a path of length at most d passing through only loyal generals will obey the same order.
2. If the commander is loyal, then any loyal lieutenant connected to him by a path of length at most $m + d$ passing only through loyal generals will obey his order.

6. RELIABLE SYSTEMS

Other than using intrinsically reliable circuit components, the only way we know to implement a reliable computer system is to use several different "processors" to compute the same result, and then to perform a majority vote on their outputs to obtain a single value. (The voting may be performed within the system, or externally by the users of the output.) This is true whether one is implementing a reliable computer using redundant circuitry to protect against the failure of individual chips, or a ballistic missile defense system using redundant computing sites to protect against the destruction of individual sites by a nuclear attack. The only difference is in the size of the replicated "processor".

The use of majority voting to achieve reliability is based upon the assumption that all the nonfaulty processors will produce the same output. This is true so long as they all use the same input. However, any single input datum comes from a single physical component—for example, from some other circuit in the reliable computer, or from some radar site in the missile defense system—and a malfunctioning component can give different values to different processors. Moreover, different processors can get different values even from a nonfaulty input unit if they read the value while it is changing. For example, if two processors read a clock while it is advancing, then one may get the old time and the other the new time. This can only be prevented by synchronizing the reads with the advancing of the clock.

In order for majority voting to yield a reliable system, the following two conditions should be satisfied:

1. All nonfaulty processors must use the same input value (so they produce the same output).
2. If the input unit is nonfaulty, then all nonfaulty processes use the value it provides as input (so they produce the correct output).

These are just our interactive consistency conditions IC1 and IC2, where the "commander" is the unit generating the input, the "lieutenants" are the processors, and "loyal" means nonfaulty.

It is tempting to try to circumvent the problem with a "hardware" solution. For example, one might try to insure that all processors obtain the same input value by having them all read it from the same wire. However, a faulty input unit could send a marginal signal along the wire—a signal that can be interpreted by some processors as a 0 and by others as a 1. There is no way to guarantee that different processors will get the same value from a possibly faulty input device except by having the processors communicate among themselves to solve the Byzantine Generals Problem.

Of course, a faulty input device may provide meaningless input values. All that a Byzantine Generals solution can do is guarantee that all processors use the same input value. If the input is an important one, then there should be several separate input devices providing redundant values. For example, there should be redundant radars as well as redundant processing sites in a missile defense system. However, redundant inputs cannot achieve reliability; it is still necessary to insure that the nonfaulty processors use the redundant data to produce the same output.

In case the input device is nonfaulty but gives different values because it is read while its value is changing, we still want the nonfaulty processors to obtain a reasonable input value. It can be shown that, if the functions *majority* and *choice* are taken to be the median functions, then our algorithms have the property that the value obtained by the nonfaulty processors lies within the range of values provided by the input unit. Thus, the nonfaulty processors will obtain a reasonable value so long as the input unit produces a reasonable range of values.

We have given several solutions, but they have been stated in terms of Byzantine generals rather than in terms of computing systems. We now examine how these solutions can be applied to reliable computing systems. Of course, there is no problem implementing a "general's" algorithm with a processor. The problems lie in implementing a message passing system that meets assumptions A1–A3 (assumptions A1–A4 for Algorithm SM(m)). We now consider these assumptions in order.

A1. Assumption A1 states that every message sent by a nonfaulty processor is delivered correctly. In real systems, communication lines can fail. For the oral message algorithms OM(m) and OM(m, p), the failure of the communication line joining two processors is indistinguishable from the failure of one of the processors. Hence, we can only guarantee that these algorithms will work in the presence of up to m failures, be they processor or communication line failures. (Of course, the failure of several communication lines attached to the same processor is equivalent to a single processor failure.) If we assume that a failed communication line cannot result in the forgery of a signed message—an assumption which we will see below is quite reasonable, then our signed message algorithm SM(m) is insensitive to communication line failure. More precisely, Theorem 4 remains valid even with communication line failure. A failed communication line has the same effect as simply removing the communication line—it lowers the connectivity of the processor graph.

A2. Assumption A2 states that a processor can determine the originator of any message that it received. What is actually necessary is that a faulty processor not be able to impersonate a nonfaulty one. In practice, this means that interprocess communication be over fixed lines rather than through some message switching network. (If a switching network is used, then faulty network nodes must be considered, and the Byzantine Generals Problem appears again.) Note that assumption A2 is not needed if A4 is assumed and all messages are signed, since impersonation of another processor would imply forging its messages.

A3. Assumption A3 requires that the absence of a message can be detected. The absence of a message can only be detected by its failure to arrive within some fixed length of time—in other words, by the use of some time-out convention. The use of time-out to satisfy A3 requires two assumptions:

1. There is a fixed maximum time needed for the generation and transmission of a message.
2. The sender and receiver have clocks that are synchronized to within some fixed maximum error.

The need for the first assumption is fairly obvious, since the receiver must know

how long he needs to wait for the message to arrive. (The generation time is how long it takes the processor to send the message after receiving all the input necessary to generate it.) The need for the second assumption is less obvious. However, it can be shown that either this assumption or an equivalent one is necessary to solve the Byzantine Generals Problem. More precisely, suppose that we allow algorithms in which the generals take action only in the following circumstances:

1. At some fixed initial time (the same for all generals).
2. Upon the receipt of a message.
3. When a randomly chosen length of time has elapsed. (I.e., a general can set a timer to a random value and act when the timer goes off.)

(This yields the most general class of algorithms we can envision which does not allow the construction of synchronized clocks.) It can be shown that no such algorithm can solve the Byzantine Generals Problem if messages can be transmitted arbitrarily quickly, even if there is an upper bound on message transmission delay. Moreover, no solution is possible even if we restrict the traitors so that the only incorrect behavior they are permitted is the failure to send a message. The proof of this result is beyond the scope of this paper. Note that placing a lower as well as an upper bound on transmission delay allows processors to implement clocks by sending messages back and forth.

The above two assumptions make it easy to detect unsent messages. Let μ be the maximum message generation and transmission delay, and assume the nonfaulty processors have clocks that differ from one another by at most τ at any time. Then any message that a nonfaulty process should begin to generate by time T on its clock will arrive at its destination by time $T + \mu + \tau$ on the receiver's clock. Hence, if the receiver has not received the message by that time, then it may assume that it was not sent. (If it arrives later, then the sender must be faulty, so the correctness of our algorithms does not depend upon the message being sent.) By fixing the time at which the input processor sends its value, one can calculate until what time on its own clock a processor must wait for each message. For example, in Algorithm SM(m) a processor must wait until time $T_0 + k(\mu + \tau)$ for any message having k signatures, where T_0 is the time (on his clock) at which the commander starts executing the algorithm.

No two clocks run at precisely the same rate, so no matter how accurately the processors' clocks are synchronized initially, they will eventually drift arbitrarily far apart unless they are periodically resynchronized. We therefore have the problem of keeping the processors' clocks all synchronized to within some fixed amount, even if some of the processors are faulty. This is as difficult a problem as the Byzantine Generals Problem itself. Solutions to the clock synchronization problem exist which are closely related to our Byzantine Generals solutions. They will be described in a future paper.

A4. Assumption A4 requires that processors be able to sign their messages in such a way that a nonfaulty processor's signature cannot be forged. A signature is a piece of redundant information $S_i(M)$ generated by process i from a data item M. A message signed by i consists of a pair $(M, S_i(M))$. To meet parts (a)

and (b) of A4, the function S_i must have the following two properties:

(a) If processor i is nonfaulty, then no faulty processor can generate $S_i(M)$.
(b) Given M and X, any process can determine if X equals $S_i(M)$.

Property (a) can never be guaranteed, since $S_i(M)$ is just a data item, and a faulty processor could generate any data item. However, we can make the probability of its violation as small as we wish, thereby making the system as reliable as we wish. How this is done depends upon the type of faults we expect to encounter. There are two cases of interest:

1. *Random Malfunction.* By making S_i a suitably "randomizing" function, we can make the probability that a random malfunction in a processor generates a correct signature essentially equal to the probability of its doing so through a random choice procedure—that is, the reciprocal of the number of possible signatures. The following is one method for doing this. Assume that messages are encoded as positive integers less than P, where P is a power of two. Let $S_i(M)$ equal $M * K_i \bmod P$, where K_i is a randomly chosen odd number less than P. Letting K_i^{-1} be the unique number less than P such that $K_i * K_i^{-1} \equiv 1 \bmod P$, a process can check that $X = S_i(M)$ by testing that $M \equiv X * K_i^{-1} \bmod P$. If another processor does not have K_i in its memory, then the probability of its generating the correct signature $M * K_i$ for a single (nonzero) message M should be $1/P$: its probability of doing so by random choice. (Note that if the processor could obtain K_i by some simple procedure, then there might be a larger probability of a faulty processor j forging i's signature by substituting K_i for K_j when trying to compute $S_j(M)$.)

2. *Malicious Intelligence.* If the faulty processor is being guided by a malicious intelligence—for example, if it is a perfectly good processor being operated by a human who is trying to disrupt the system—then the construction of the signature function S_i becomes a cryptography problem. We refer the reader to [1] and [4] for a discussion of how this problem can be solved.

Note that it is easy to generate the signature $S_i(M)$ if the process has already seen that signature. Hence, it is important that the same message never have to be signed twice. This means that, when using SM(m) repeatedly to distribute a sequence of values, sequence numbers should be appended to the values to guarantee uniqueness.

7. CONCLUSION

We have presented several solutions to the Byzantine Generals Problem, under various hypotheses, and shown how they can be used in implementing reliable computer systems. These solutions are expensive in both the amount of time and the number of messages required. Algorithms OM(m) and SM(m) both require message paths of length up to $m + 1$. In other words, each lieutenant may have to wait for messages that originated at the commander and were then relayed via m other lieutenants. Fischer and Lynch have shown that this must be true for any solution that can cope with m traitors, so our solutions are optimal in that respect. Our algorithms for a graph that is not completely connected require

message paths of length up to $m + d$, where d is the diameter of the subgraph of loyal generals. We suspect that this is also optimal.

Algorithms OM(m) and SM(m) involve sending up to $(n - 1)(n - 2) \cdots (n - m - 1)$ messages. The number of separate messages required can certainly be reduced by combining messages. It may also be possible to reduce the amount of information transferred, but this has not been studied in detail. However, we expect that a large number of messages will still be required.

Achieving reliability in the face of arbitrary malfunctioning is a difficult problem, and its solution seems to be inherently expensive. The only way to reduce the cost is to make assumptions about the type of failure that may occur. For example, it is often assumed that a computer may fail to respond but will never respond incorrectly. However, when extremely high reliability is required, such assumptions cannot be made, and the full expense of a Byzantine Generals solution is required.

REFERENCES

1. DIFFIE, W., AND HELLMAN, M.E. New directions in cryptography. *IEEE Trans. Inf. Theory IT-22* (Nov. 1976), 644–654.
2. DOLEV, D. The Byzantine generals strike again. *J. Algorithms 3*, 1 (Jan. 1982).
3. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.
4. RIVEST, R.L., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (Feb. 1978), 120–126.

Received April 1980; revised November 1981; accepted November 1981