12. The Byzantine Generals

DANNY DOLEV, LESLIE LAMPORT, MARSHALL PEASE,

and

ROBERT SHOSTAK

ABSTRACT

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. The solution for a general distributed system requires connectivity of more than twice the number of traitors, while in the case of unforgeable written messages, connectivity larger than the number of traitors suffices. Applications of the solutions to reliable computer systems are then discussed.

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

This work was supported in part by the National Aeronautics and Space Administration under contract number NAS1-15428 Mod. 3, and the Ballistic Missile Defense Systems Command under contract number DASG60-78-C-0046, and the Army Research Office under contract number DAAG29-79-C-0102.

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

CONDITION 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:

CONDITION 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 will 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 that 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.

CONDITION 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:

CONDITION 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):

CONDITION 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:

CONDITION IC1. All loyal lieutenants obey the same order.
CONDITION 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 that 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 will consider signed, written messages, for which this is not true.

We now study 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 Fig. 12.1, 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 Condition IC2 to be satisfied, Lieutenant 1 must obey the order to attack.

Now consider another scenario, shown in Fig. 12.2, 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 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.

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

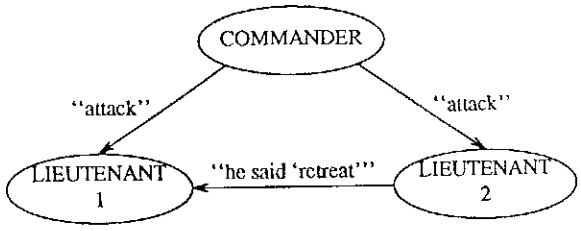

Fig. 12.1. Lieutenant 2 a traitor.

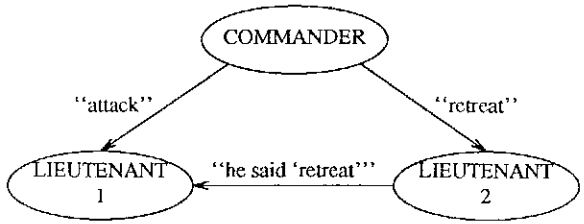

Fig. 12.2. The commander a traitor.

that the commander said "attack." Therefore, in the scenario of Fig. 12.2, 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 reference 12.

Using this result, we can show that no solution with fewer than $3m + 1$ generals can cope with m traitors.* The proof is by contradiction—we assume such a solution for a group of $3m$ or fewer generals, 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 will call the generals of the assumed solution Albanian generals, and those of the constructed solution will be called Byzantine generals. Thus, starting from an algorithm that allows $3m$ or fewer Albanian generals to cope with m traitors, we will 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

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

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:

CONDITION IC1′. All loyal lieutenants attack within ten minutes of one another.

CONDITION IC2′. If the commanding general is loyal, then every loyal lieutenant attacks within ten 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 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 retreat 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:
   If the time is 1:10 or earlier, then attack.
   If the time is 1:50 or later, then retreat.
   Otherwise, continue to Step 2.

2. Ask the other lieutenant what decision he reached in Step 1.
   If the other lieutenant reached a decision, then make the same decision he did.
   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. If follows from IC1′ that, if one lieutenant decided to attack in Step 1, then the other cannot decide to retreat in Step 1. Hence, they will both either 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 have shown 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. Note that assumptions A1–A3 do not imply that a general hears any message sent between two other generals.

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 Oral Message algorithms as OM(m) for all nonnegative integers m, by which a commander sends an order to $n - 1$ lieutenants. We will show that OM(m) solves the Byzantine Generals Problem for $3m + 1$ or more generals in the presence of at most m traitors. We will 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* $(v_1, \cdots, v_{n-1})$ equals v. (Actually, it assumes a sequence of such functions—one for each n.) There are two natural choices for the value of *majority* $(v_1, \cdots, v_{n-1})$:

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* $(v_1, \cdots, v_{n-1})$.

To execute 3, every processor must know when to apply the majority function, in other words, when to stop waiting for more values to come. To do this, one can use some sort of time-out technique, as we will discuss in Section 6. Note that recently, Fischer, Lynch, and Paterson[8] proved that there is no way to reach any agreement unless we assume some bound on the time at which a reliable processor responds.

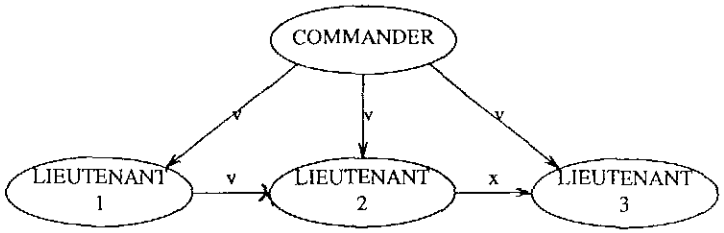

Fig. 12.3. Algorithm OM(1)—Lieutenant 3 a traitor.

To understand how Algorithm OM(m) works, we consider the case $m = 1$, $n = 4$. Figure 12.3 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 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 $(n - 1)$, . . . , $(n - k)$ times to send a value prefixed by a sequence of k lieutenant's numbers. This implies that the algorithm requires sending an exponential number of messages. There exist algorithms which require only a polynomial number of messages,[4] but they are substantially more complex than the one we present.

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 Condition IC2 if there are more than $2k + m$ generals, and at most k traitors.

Proof: The proof is by induction on m. Condition IC2 only specifies what must happen if the commander is loyal. Using A1, it is easy to see the trivial algorithm OM(0) works if the commander is loyal, so the lemma is true $m =$

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 $v_1, \cdots, v_{n-1}$, and therefore obtain the same value *majority* $(v_1, \cdots, v_{n-1})$ in Step 3, proving IC1.

4. A SOLUTION WITH SIGNED MESSAGES

As we saw from the scenario of Fig. 12.1 and 12.2, 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.

Having 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 lieutentants, 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 uses a function *choice*, which is applied to a set of orders to obtain a single one. It is defined as follows:

If the set V consists of the single element v,
then *choice* $(V) = v$,
otherwise *choice* (V) = **RETREAT**

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, containing 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. We assume the existence of a bound on the time it takes correct processors to sign and relay a message. Thus, it implies the existence of some phases such that, if a message with r signatures arrives after phase r, then only faulty processors relayed it, so it can be ignored. This assumption does not necessarily mean complete synchronization of the processors.

Algorithm SM(m)
Initially $V_i = \phi$.

1. The commander signs and sends his value to every lieutenant at phase 0.
2. For each i:
   A. If Lieutenant i receives a message of the form $v{:}0$ from the commander at phase 0, 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.
   B. If Lieutenant i receives a message of the form $v{:}0{:}j_1{:}\cdots{:}j_k$ at k, $1 \leq k \leq m$, V_i contains at most one value, v is not in the set V_i, and the signatures belong to the different lieutenants, 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 $j_1, \cdots, j_k$.

3. For each i: At the end of phase m he obeys the order *choice* (V_i).

Observe that the algorithm requires $m + 1$ phases of message exchange. Note that in Step 2, Lieutenant i ignores any message containing an order v that is already in the set V_i, and accepts at most two different orders originated by the commander.

Moreover, Lieutenant i ignores any messages that do not have the proper form of a value followed by a string of different 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 $(n - k - 2)(n - k - 3), \ldots (n - m - 2)$ copies of the message if it has been signed by k lieutenants.)

Figure 12.5 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

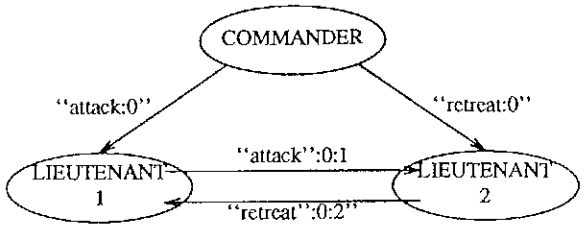

Fig. 12.5. Algorithm SM(1)—The commander a traitor.

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 Fig. 12.2, 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 on time in Step 2A. 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 2B. 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 function choice applied to the sets of orders V_i and V_j that they receive in Step 2 induces the same value. Therefore, to prove IC1 it suffices to prove two parts: one, if a loyal lieutenant i puts exactly one order v into V_j in Step 2, then every loyal lieutenant will put exactly the same order v into V_j in Step 2; two, if V_j has two elements for some loyal lieutenant j, then V_k has two elements for any other loyal lieutenant k.

To prove the first part, we must show that j receives a properly signed message containing that order. If i receives the order v in Step 2A on time, then he sends it to j in Step 2A(ii), so that j receives it on time (by A1). If i adds the order to V_i in Step 2B, 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 $j_1, \cdots, j_m$ is loyal. This loyal lieutenant must have sent j the value v when he first received it, so j must, therefore, receive that value.

Similar arguments prove that if any loyal lieutenant i decides to put two orders in V_i, then every other loyal lieutenant will decide to do so.

This completes the proof.

During the algorithm, every loyal lieutenant relays to every other lieutenant at most two orders. Therefore, the total number of messages exchanged is bounded by $2n(n-1)$, where n is the total number of generals. By using more phases and more sophisticated algorithms, one can reduce the total number of messages to $O(n + m^2)$ as shown in reference 5.

5. MISSING COMMUNICATION PATHS

Thus far, we have assumed that a general (or lieutenant) can send messages directly to every other general (or lieutenant). We now remove this assumption. Instead, we supposed that physical barriers place some restrictions on who can send messages to whom. We consider the generals to form the nodes of a simple,* finite, undirected network graph G, where an arc between two nodes indicates that those two generals can send messages directly to one another. We now extend algorithms OM(m) and SM(m), which assumed G to be completely connected, to more general graphs.

The commander sends his value through routes in the network. For simplicity, assume that every message contains the information about the route through which it is supposed to be delivered. Thus, before sending a message, the commander chooses a route and sends the message containing the route. The receiving lieutenant, however, does not know in advance the route through which it is going to receive the message. Notice that a traitor may also change the routing through which the message is supposed to be delivered. Moreover,

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

a traiter may also produce many false copies of the message it is supposed to relay, then send them through various routes of its own choice.

A traitor may change the record of the route to prevent the receiving lieutenant identifying it as the source of faulty messages. To ensure the inclusion of traitors' names in the routes, assume that, after a loyal lieutenant receives a message to relay, he makes sure the lieutenant from which the message has arrived is supposed to relay it to him. Only then does he relay the message to the next lieutenant along the route to the receiving lieutenant.

A network has *connectivity* k if, for every pair of nodes, there exists k node-independent paths connecting them.

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: Let $\{a_1, \cdots, a_r\}$ be the set of copies of the commander's value received by Lieutenant i. Let U_i be a set of lieutenants that does not contain the commander himself. A set U_i is called a set of *suspicious* lieutenants determined by lieutenant i if every message a_j that did not pass through lieutenants in U_i carries the same value.

Algorithm Purifying (m, a_1, $\cdots$, a_r, i)

1. If a set U_i of up to m suspicious generals exists, then the *purified* value is the value of the messages that did not pass thorugh U_i. If no message is left, the value is **RETREAT.**
2. If there is no set U_i of cardinality up to m, then the purified vlaue is **RETREAT**.

Notice that if more than one set of suspicious generals exists, then there may be many purified values, but because of the way the algorithm will be used, a plurality of possible values will pose no problem. Before proving that the Purifying Algorithm actually does the right filtration, consider application of the Purifying Algorithm to the network shown in Fig. 12.6.

The network contains 10 generals, and at most 2 traitors. Assume that s and u are the faulty generals. The commander s sends the value a to Lieutenants 1 and 2, and the value b to the other lieutenants. Assume that Lieutenant 1 receives s's value through the following paths:

1. a: $s1$
2. a: $s21$
3. a: $su1$
4. b: $s741$
5. b: $s851$.

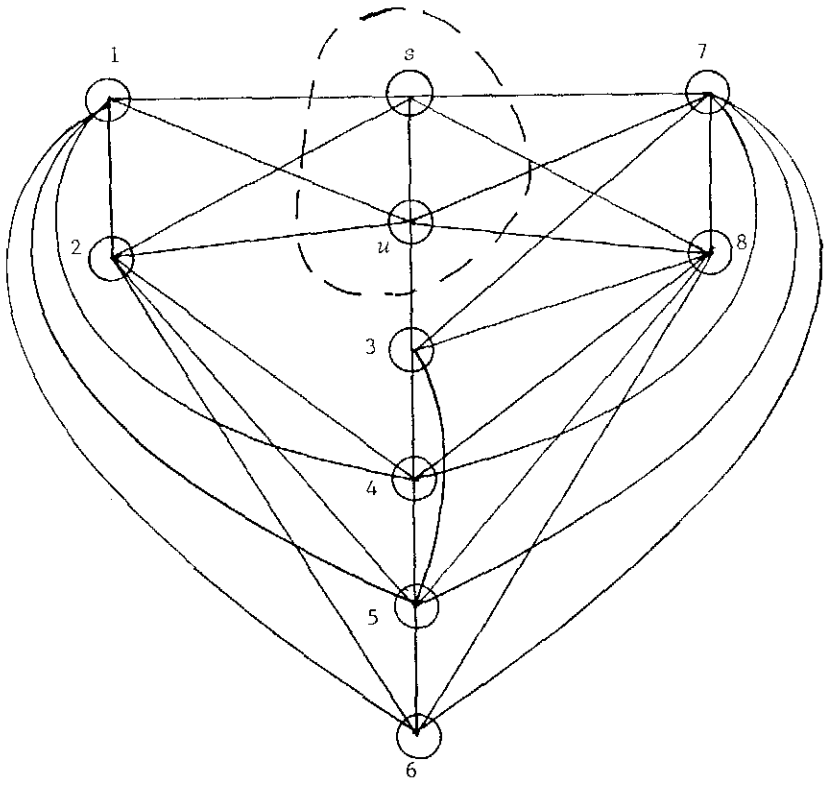

Fig. 12.6. Ten generals with two traitors, s is the commander.

The Purifying Algorithm provides the purified value a to Lieutenant 1, by choosing $\{7, 8\}$ as the set of suspicious generals. Similarly, Lieutenant 2 obtains the value a. But the rest of the network obtain the value b by choosing $\{1, 2\}$ as the set of suspicious generals.

The following theorem proves that, with sufficient connectivity, all of the loyal lieutenants obtain the same value if the commander is loyal.

Theorem 3: Let G be a network of generals which contains at most t traitors, and the connectivity of which is at least $2m + 1$. If a loyal commander sends $2m + 1$ copies of its value to every lieutenant, through disjoint paths, then, by use of the Purifying Algorithm, every loyal lieutenant can obtain the commander's value.

Proof: The loyal commander sends every lieutenant $2m + 1$ copies of a value, through disjoint paths. It sends the same value to all lieutenants. Let a_1,

$\cdots$, a_r be the set of all of the copies of the commander's value that Lieutenant i receives. There are at most m traitors; therefore, at most m values might be lost. This implies that the number of copies, r, is at least $m + 1$. At least $m + 1$ of the messages are relayed through routes which contain only loyal generals; each one of the loyal lieutenants relays the message faithfully without changing it. This implies that at least $m + 1$ of the received copies carry the original value. Note that, if the commander were a traitor, then the above reasoning would fail to hold.

It may be that the number of copies received is much more than $m + 1$, and even that the majority of them carry a faulty value. The task of Lieutenant i is to find the correct value out of this mess. It does this by applying the Purifying Algorithm. Observe that the technique, described at the beginning of the Section, of adding the names of the generals along the route to the message, enables i to differentiate among the values. Every message which passed through traitors contains at least one name of a traitor; more precisely, every list of generals added to a message contains at least the name of the last traitor that relayed it.

Step 1 of the Purifying Algorithm requires one to look for a set U_i of up to m generals with the property that all of the values which have not been relayed by generals from this set are the same. The network contains at most m traitor generals, and by assumption, the commander is loyal. Therefore, Lieutenant i should be able to find such a set U_i; it may be that the set he finds is not exactly the set of traitors, but U_i necessarily eliminates the wrong values. The set U_i cannot eliminate the correct values, because there are at least $m + 1$ independent copies of them and U_i can eliminate at most m independent copies. This completes the proof of the theorem.

In the case where the commander is a traitor, Theorem 3 does not ensure the ability to reach a unique agreement on a value. But the way we will use it in algorithm OM(m) will overcome the faultiness of the commander.

To obtain Byzantine Agreement in a network with connectivity k, $k \geq 2m + 1$, we improve algorithm OM(m) as follows: whenever a general sends a message to another, he sends it through $2m + 1$ disjoint paths; whenever a lieutenant has to receive a message, he uses the Purifying Algorithm to decide on a purified value. Call the improved algorithm OM$'$(m).

To prove the validity of the algorithm OM$'$(m), observe that the same general can be used again and again as a relay in the disjoint paths between pairs of generals, even if he was a commander in previous recursions. Moreover, even being a traitor does not matter for the simple reason that the total number of independent paths that would be affected by traitors will never exceed m.

Theorem 4. Let G be a network of n generals with connectivity $k \geq 2m + 1$, where $n \geq 2m + 1$. If there are at most m traitors, then Algorithm OM$'$(m) (with the above modifications) solves the Byzantine Generals Problem.

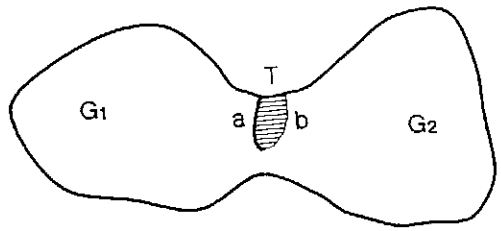

Fig. 12.7. T is the set of traitors.

Proof. The proof is essentially the same as the proof of Theorem 1, using Theorem 3 everywhere to show that, when a loyal lieutenant sends a value, every other loyal lieutenant agrees on it. The fact that we use the whole network to relay the information again and again eliminates any loss of connectivity, and enables us to obtain the desired result. The details are left to the reader.

To show that Theorem 4 is the best possible, we prove that the connectivity of $2m + 1$ is necessary for solving the Byzantine General Problem.

The case in which the number of traitors is not less than half of the connectivity is easier to visualize, and is proved in Lemma 2. Figure 12.7 describes the case schematically. The basic idea is that, if the traitors are not less than half of the bottleneck, then they can prevent the loyal generals from reaching an agreement by behaving as a filter. Every message that passes from right to left would be changed to carry one value, and every message in the reverse direction would carry another value. This behavior can cause all of the generals on the right side to agree on a different value from those on the left side.

Lemma 2: The Byzantine General Problem cannot be solved in a network of n generals if the number of traitors is not less than half the connectivity of the network.

Proof: Let G be a network with connectivity k, and let $s_1, \cdots, s_k$ be a set of generals which disconnect the network into two non-empty parts, G_1 and G_2. Assume that the subset $s_1, \cdots, s_m$ is the set of traitors, where $m \geq k/2$. Consider the following cases for the various locations in which the commander can be.

Assume the commander s is in the subnetwork G_1, and that he sends the value a to all of the lieutenants in the network. The traitors can follow the doctrine: change every message which passes from G_1 to G_2 to carry the value b; and leave every other value as a. Change the messages passing back from G_2 to G_1 to carry the value a. In this situation, every lieutenant in G_1 can consider s to be a loyal general, and thus agree on a. Similarly, the processors $s_{m+1}, \cdots, s_k$ choose a. But every receiver in G_2 cannot consider s a traitor.

They are able to ignore the conflicting values they have received by ignoring either the set $s_1, \cdots, s_m$ or $s_{m+1}, \cdots, s_k$. On the other hand, they cannot agree on a value, because each of the values can be correct, depending upon what the commander has said and which generals are traitors. Since $m \geq k$, the lieutenants in G_2 will choose b, in contradiction to IC2. The case where the commander is in G_2 is identical by symmetry.

Assume now that the commander is in the set $s_1, \cdots, s_k$. If the commander is loyal and sends the same value a to every lieutenant, then by reasoning, similar to the previous case, the traitors can prevent agreement. If the commander is a traitor, he can send the value a to G_1 and b to G_2. Thus, similarly to the previous case every decision implies violation of IC2. For a more rigorous proof see reference 3.

Our extension of Algorithm OM(m) requires that the graph G be $2m + 1$ connected, which is a rather strong connectivity hypothesis. 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 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 subnetwork formed by the loyal generals be connected. We will 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 2B, lieutenant i only sends the message to every neighboring lieutenant not among the j_r.

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

Theorem 5: For any m and d, if there are at most m traitors and the network 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 will just be sketched. 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 that the commander is a traitor and must show that all loyal lieutenants have received a unique order, or every one decides on **RETREAT**. The idea is exactly as above. 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 network of loyal generals is connected, then SM($n - 2$) (as modified above) solves the Byzantine Generals algorithm for n generals.

Proof: Let d be the diameter of the network 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 5 assumes that the subnetwork 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; and 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 for implementing a reliable computer system is to use several different "processors" to compute the same result, and 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—e.g., 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 that they produce the same output); and 2) If the input unit is nonfaulty, then all nonfaulty processes use the value it provides as input (so that 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.

REFERENCES

1. DeMillo, R. A., N. A. Lynch, and M. Merritt, Cryptographic protocols, in *Proc. 14th ACM SIGACT Symp. on Theory of Computing*, pp. 383–400, May 1982.
2. Diffie, W. and M. E. Hellman, *New directions in cryptography*, *IEEE Trans. Inform. Theory IT-22*, pp. 644–654, Nov. 1976.
3. Dolev, D. The Byzantine generals strike again, *J. Algorithms*, vol. 3, pp. 14–30, Jan. 1982.
4. Dolev, D., M. Fischer, R. Fowler, N. Lynch, and R. Strong, Efficient Byzantine agreement without authentication, *Info. and Control*, vol. 3, pp. 257–274, 1983.
5. Dolev, D. and R. Reischuk, Bounds on information exchange for Byzantine agreement, *JACM*, vol. 32, pp. 191–204, 1985.
6. Dolev, D., R. Reischuk, and H. R. Strong, 'Eventual' is earlier than 'immediate,' *Proc. 23rd Annual IEEE Symp. on Foundations of Computer Science*, pp. 196–203, 1982.
7. Dolev, D. and H. R. Strong, Authenticated algorithms for Byzantine agreement, *SIAM J. on Comp.*, vol. 12, pp. 656–666, 1983.
8. Fischer, M., N. Lynch, and M. Paterson, Impossibility of distributed consensus with one faulty processor, *JACM*, vol. 32, pp. 374–382, 1985.
9. Lamport, L. and P. M. Melliar-Smith, *Synchronizing Clocks in the Presence of Faults*, Tech. Rep., Computer Science Lab., SRI International, 1984.
10. Lamport, L., R. Shostak, and M. Pease, The Byzantine generals problem, *ACM Trans. on Programming Languages and Systems*, vol. 4, pp. 382–401, July 1982.
11. Lynch, N. and M. Fischer, A lower bound for the time to assure interactive consistency, *Information Processing Letters*, vol. 14, pp. 182–186, 1982.
12. Pease, M., R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *J. ACM 27*, vol. 2, pp. 228–234, Apr. 1980.
13. Rivest, R. L., A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *CACM*, vol. 21, pp. 120–126, Feb. 1978.

Chapters # 1

CONCURRENCY CONTROL AND RELIABILITY IN DISTRIBUTED SYSTEMS

Edited by

Bharat K. Bhargava
Department of Computer Science
Purdue University
West Lafayette, Indiana

VNR VAN NOSTRAND REINHOLD COMPANY
New York