RANDOMIZED BYZANTINE GENERALS

Michael O. Rabin*

Hebrew University and Harvard University

Abstract. We present a randomized solution for the Byzantine Generals Problems. The solution works in the synchronous as well as the asynchronous case and produces Byzantine Agreement within a fixed small expected number of computational rounds, independent of the number n of processes and the bound t on the number of faulty processes. The solution uses A. Shamir's method for sharing secrets. It specializes to produce a simple solution for the Distributed Commit problem.

0. INTRODUCTION

We present a randomized solution for the well known Byzantine Generals (BG) problem. This problem was introduced by Lamport, Pease and Shostak [LPS80], and was already the subject of numerous interesting studies. In particular, Dolev and Strong [DS81], have shown, extending a result of Lynch and Fischer [LF82], that for n processes with up to t faulty ones, $t+1$ computing phases are necessary for reaching Byzantine Agreement. Another result [FLP82] shows that the BG problem has no solution in the asynchronous case.

Our randomized protocol is certain to achieve Byzantine Agreement, and the expected number of rounds required to do so is four, independent of n and t. The total expected number of messages exchanged is cnt, where c is a small constant.

Another version of the protocol, employs a fixed number R of rounds to reach Byzantine agreement, but there is a probability 2^{-R} of error.

The algorithms we use are randomized in the sense of [Ra76], i.e., they employ randomly chosen numbers. Their properties and efficacy do not depend on an assumption of random behavior of the faulty processes.

*Research supported in part by NSF Grant MCS-8121431.

Like some other solutions, see [DS81], our solution employs authentication of messages by digital signatures. This requires that the processes in the system be supplied in advance by a non-faulty Dealer, with a directory of public-keys.

A salient novel feature is the use of randomly chosen secrets which are shared by the processes in the manner invented by A. Shamir [Sh79]. This also requires that the processes be supplied in advance with certain information by a non-faulty Dealer.

The solution applies, with appropriate modifications, to the synchronized version as well as to the completely asynchronous version of the problem.

A different randomized solution for the BG problem was given, independently, by Ben-Or [BO83]. Ben-Or treats the case of a two-valued initial message, and the processes autonomously toss a coin until a large number of the individual outcomes coincide. This solution requires, in general, a number of rounds and messages exponential in n.

If we restrict, in our solution, the modes of failure to just breakdowns of processes, we can dispense with authentication and with Shamir's method for sharing a secret. We then get a very simple and robust solution for the Distributed Commit problem.

Thus, in addition to being quite simple, our solution has stronger properties than other existing solutions. It may well lend itself to practical implementation. The author would like to thank D. Dolev and H. Gaifman for many helpful conversations on the topics of this paper.

1. BASIC CONCEPTS

Let Gi, $1 \leq i \leq n$, be processes (the "generals"). Assume that every Gi can directly exchange messages with every other Gj.

0272-5428/83/0000/0403$01.00

Every Gi has a message Mi, called Gi's initial message, and they have to agree on a common value (content) for the message. To this end, each Gi executes a program Pi called the Agreement Protocol. The protocols Pi involves exchanging messages with other processes Gj, deciding what value to adopt as the common message, and deciding when to stop.

As long as a Gi computes according to Pi it is called proper. Once a process Gi deviates from Pi it is faulty, and is considered to remain faulty even if, later on, it reverts back to following Pi.

Each process Gi has a variable, message(i), local to it. This variable holds, at the end of Gi's execution of its agreement protocol, Gi's version of the message.

Definition 1. The processes in a set $\{Gi : i \in P\}$ are said to reach an agreement about the value of the message if each Gi, $i \in P$, does stop, and at the end of the execution of their protocols, message(i) = message(j) for all $i, j \in P$.

Definition 2. We say that the proper processes have reached Byzantine Agreement if

1. All the proper processes reach agreement.
2. If all proper Gi have the same initial message $Mi = M$ then the proper processes agree on M as the value of the message.

Note that when we talk about Byzantine Agreement, nothing is assumed about the faulty processes.

If all the proper processes have the same initial message then the system will be called proper, otherwise faulty.

Definition 3. Let n and $t < n$ be integers. A set of protocols (programs) Pi for the processes Gi, $1 \leq i \leq n$, is a solution for the Byzantine Generals problem for (n,t) if the following holds. Let S be a system of processes Gi computing according to Pi and assume that no more than t processes become faulty. Then whenever the processes have initial messages, the proper Gi will reach Byzantine Agreement.

A solution for the Byzantine Generals problem will be called a Byzantine Agreement Protocol (BAP).

It should be noted that in a system with up to t faulty processes, the processes to become faulty are not selected in advance. Rather, we make the worst case assumption that at any time during the computation, an adversary can select further processes which will become faulty, provided that the total number of faulty processes will not exceed t.

For randomizing protocols Pi, which employ randomly chosen numbers in the computation, we have several possible notions of BAP.

We can talk about protocols which achieve Byzantine Agreement with a small probability of error. More precisely, given $o < \varepsilon$ we say that randomizing protocols Pi, $i \leq i \leq n$, are a $1 - \varepsilon$ reliable Byzantine Agreement Protocol for (n,t), if for every fixed or randomized behavior of up to t faulty processes, the proper processes will reach Byzantine Agreement with probability at least $1 - \varepsilon$.

Another possibility is to demand that for some constant C, the proper processes achieve Byzantine Agreement within an expected number C of phases, and this without error. The notion of a computation phase will be explained shortly.

We shall present randomized BAP's of both types.

We shall present the detailed solution for the completely asynchronous case. In this situation we make the assumption that whenever a proper Gi sends a message to another proper Gj, the message will eventually reach Gj. Also, a proper process will execute the instructions of its protocol within a finite time. In fact, these requirements could be incorporated into the definition of a proper process. Throughout the following, we denote by t the upper bound on the number of processes that may become faulty.

The processes reach agreement on the common message by exchanging information. In such an exchange a process Gi will request certain information from certain k other processes. In waiting for the response, Gi should not wait for more than $k - t$ replies, because up to t processes may be faulty and never respond. So that waiting for more than $k - t$ responses could block a process. These considerations motivate the following

Definition 4. A phase in the computation of a proper Gi, is the time interval between Gi's request for information from k other processes, and receipt of at least $k - t$ replies. The computation time required by Gi, once the information is received, is also included in the phase.

The k in the above definition is determined by the current instruction executed by the protocol Pi. The actual duration of a phase may be different for different processes, and may vary from phase to phase for the same Gi. But our assumptions insure that for a proper process, each phase is finite.

We assume that each process Gi has a local phase-clock p(i), and that Pi assigns p(i) := p(i) + 1 at the end of each phase.

2. AUTHENTICATION

Our solution will use digital signatures for authentication of messages. One implementation, see [DH76], [RSA78], is to have a public directory containing for each participant B a public key K_B. When the participant B needs to authenticate a message M, he employs a secret key D_B to compute another message $\sigma_B(M) = N$. Every other user can recover M form N by use of K_B, and the fact that M was so recovered is conclusive proof that it originated with B. We omit details of the implementation that ensure that nobody other than B can produce any message of the form $\sigma_B(M)$.

For each Gi we shall denote by $\sigma_i(M)$ the message M authenticated by Gi.

In order to render it impossible for a process Gj to reuse another processes' old authenticated message $\sigma_i(M)$, we employ time-stamping. Each Gi will incorporate the current reading p(i) of its phase-clock into any message M to be authenticated, thus producing $\sigma_i(M,p(i))$. By keeping track of Gi's most recent time-stamp, a recipient can distinguish between messages newly received from Gi, and old messages. In the future we shall assume time-stamping, without explicitly mentioning it.

The public-key directory is part of the data in each Pi, and must be incorporated by a <u>non-faulty</u> "dealer" at the creation of the processes Gi.

3. LOTTERY

Our algorithm will employ a lottery procedure by which the proper Gi's can agree on a randomly chosen $s = 0,1$.

In [Sh79] A. Shamir gave a method of sharing a secret s between n participants so that every k or more cooperating participants can reconstruct s, but no fewer than k participants can do so. According to Shamir's scheme, a "dealer" who chooses the secret s to be shared, prepares a sequence $I_i(s)$, $1 \leq i \leq n$, and gives $E_i = I_i(s)$ to Gi. The E_i, and Shamir's algorithm, have the property that from every set $\{E_{i_1},\ldots,E_{i_k}\}$, the secret s can be reconstructed. But no collection of fewer than k values E_i determines s.

For our purposes we assume that the dealer is a non-faulty process D which prepares in advance the programs for the individual Gi's. The dealer D takes $k = t+1$, where t is the bound on the number of faulty processes. It <u>randomly</u> and independently chooses a sequence $s_1,\ldots,s_N$ where each $s_m = 0$ or $s_m = 1$. The dealer D then computes

$$E_i^{(m)} = (I_i(s_m),m) \quad i \leq i \leq n, \quad 1 \leq m \leq N.$$

He authenticates all the $E_i^{(m)}$ with his public-key digital signature σ_D and hands to Gi the sequence

$$\sigma_D(E_i^{(1)}),\ldots,\sigma_D(E_i^{(N)}) .$$

The number N represents the number of lottery rounds that the processes are expected to play during the duration of the system.

The lottery procedure admits a parameter $m \leq N$, so that Lottery(m) is the m-th lottery round.

When playing in Lottery(m), Gi requests from 2t other processes their $\sigma_D(E_j^{(m)})$. Since these messages are signed by D, Gi is certain of their authenticity.

Since there are at most t faulty processes who may not answer, Gi will have at least $k = t+1$ messages $(I_{j_1}(s_m),m)$, $\ldots,(I_{j_k}(s_m),m)$. The value m of the second coordinate will assure him that he is dealing with the m-th shared secret. Process Gi will now compute s_m from the $I_j(s_m)$ he has, using Shamir's method.

At the same time Gi will send his $\sigma_D(E_i^{(m)})$ to any process G that requests it.

Thus at the end of the execution of Lottery(m) by the proper Gi, at least all the proper processes will share s_m. Later on we shall explain how the proper Gi ensure that despite their asynchronous behavior, s_m is not revealed prematurely.

4. THE AGREEMENT PROTOCOL

We shall describe the BAP by a sequence of procedures written informally in Pascal style. Each process has a number of variables local to it. By the notation v(i) we mean that v(i) is local to Gi, but is global to Gi's protocol Pi.

In particular p(i) will denote Gi's current phase-clock reading. The variable message(i) will hold at any time Gi's current version of the message, which will be one of the initial messages Mj or else the value "system faulty".

The overall structure of BAP will consist of Polling, where Gi polls the other processes on their value of the message. Lottery, where the Gi decide on a common random bit s. And Decision, where Gi determines, using s, whether to adopt the plurality candidate version of the message obtained through Polling, as his current version of the message. This is repeated a fixed number R of times. The value of R is determined by the desired realiability, which will turn out to be $1-2^{-R}$. Thus the BAP will be

```
Procedure BAP; {for Gi}

  begin
    message(i) := Mi; {Gi's initial message}
    For k = 1 to k = R do {k local to Pi}
    begin
      Polling; {k is a parameter of Polling}
      Lottery; {k is a parameter of Lottery}
      Decision
    end {For}
  end; {BAP}
```

```
Procedure Polling;

  begin
    send σi(message(i),k) to all; {σi is
    Gi's authentication. The k indicates
    to recipients the version-number.}
    request the k-th version of message(j)
    from all;
    collect incoming σj(message(j),k),
    one from each responding Gj, until
    n - t values received; {including own
    value}
    temp(i) := that M which occurred most
               often among the received
               (message(j),k); {the plurality
               candidate}
   count(i) := |{j:(message(j),k) =
               (temp(i),k)}|
               {count of multiplicity of
                most popular k-th version}
  end; {Polling}
```

<u>Remark</u>. Strictly speaking, the collection <u>by Gi</u> of incoming communications σ_j(message(j),k) is not done just during the k-th iteration of Polling. Since the system S is asynchronous, some early Gj may send its k-th communication to Gi while the latter is in an earlier iteration of Polling. Thus Pi should incorporate a listening process Li which runs parallel to all other procedures of Pi, and stores all incoming communications from the other Gj as they arrive. The procedures Polling and Lottery collect the appropriate messages from Li.

We come now to the heart of the randomizing BAP. At the end of Polling every proper Gi has a value temp(i) which from <u>his</u> point of view is the plurality candidate for the common message, and a value count(i) giving the number of times Gi received the message temp(i).

We shall see later that for $t < n/4$, if the system is proper, i.e. the initial message of every proper process was M, then for every proper Gi we shall have temp(i) = M and $n-2t \leq$ count(i), after every execution of Polling.

If, however, the system is faulty, then the up to t faulty Gj, can cause the different proper Gi to have different values for temp(i), and force count(i) to almost any value in [1,n].

We must provide Gi with a rule for deciding whether to adopt temp(i) as the next candidate for value of the common message, or whether the system is faulty. A fixed rule such as $n/2 \leq$ count(i) is not appropriate, because if the system is faulty then the faulty Gj can arrange it so that for some proper Gi $n/2 \leq$ count(i), and for others count(i) $< n/2$. This would foil agreement.

The solution is to base everybody's decision on a comparison $n/2 \leq$ count(i), or $n-2t \leq$ count(i), depending on whether a randomly chosen s, which is not known to the faulty Gj before the end of Polling by at least one proper process, has value 0 or 1.

```
Procedures Lottery; {for Gi}

  begin
    ask for Ej^(k) from all;
    send Ei^(k) to all;
    wait until t different values of Ej^(k)
      have arrived;
    compute s_k from Ei^(k) and the available
      Ej^(k);
  end; {Lottery}
```

Next we give the procedure for deciding whether to adopt the value temp(i) resulting from Polling as the new (but not necessarily final) candidate for the value of the message.

```
Procedure  Decision; {for Gi}

  begin
    s := s_k; {s_k is the current secret
              for Gi}
    If (s=0 and n/2 ≤ count(i)) or (s=1 and
              n - 2t ≤ count(i)) then
              message(i) := temp(i)
    else
      message(i) := "system faulty"
  end; {Decision}
```

5. CORRECTNESS

Theorem 1. Let $t < n/10$. All the proper processes will end their execution of BAP. If the system is proper and the initial message of every proper process is M, then at the end of BAP we shall have message(i) = M for all proper Gi.

If the system is faulty then at the end of BAP, with probability at least $1-2^{-R}$, all proper processes Gi will have the same value for message(i).

Proof. We omit the proof that every proper process will indeed stop. I.e. we prove fully only partial correctness.

Observe that if the proper processes (asynchronously) enter the k-th iteration of the For statement of BAP in a state where all have the same value V for their k-th version of message(i), then they will end that iteration with the value message(i) = V.

This is true because at the end of Polling, every proper Gi will have temp(i) = V and $n - 2t \leq$ count(i). Hence irrespective of whether $s = 0$ or $s = 1$, Decision will assign message(i) := temp(i).

Thus, once the proper Gi reach the above state, the common value of message(i) will stay unchanged to the end of the For statement, and the proper Gi's will have reached agreement.

In particular, if the system is proper and its initial message is M for all proper processes, then all proper Gi will have message(i) = M at the end of BAP.

Consider the first proper process, say it is Gi, to have finished its k-th iteration of Polling. At that time Gi has accumulated n-t responses of the form σ_j(message(j),k). Of these, at least n-2t are from proper processes. W.l.o.g., let these be the processes $\{G_1,\ldots,G(n-2t)\} = H$. Only then does Gi releases its piece $E_i^{(k)}$ of the k-th secret s_k. So the faulty processes can have s_k only after the k-th version of message(j) has been determined by the $Gj \in H$.

First assume that there exists a V so that for at least $n - 4t$ (proper) processes $Gj \in H$, the k-th version is message(j) = V. Let now Gm be any proper process. At most t responses σ_j(message(j),k) from proper processes will not reach Gm by the time it finished the k-th iteration of Polling. Therefore, in its k-th round of Polling it receives responses from at least $n - 5t$ of the above mentioned processes, and all these are of the form $\sigma_j(V,k)$. Since $n/2 < n-5t$, the value V is the majority value. Hence temp(m) = V and $n/2 < n - 5t \leq$ count(m) at the end of the k-th invocation of Polling, for every proper Gm.

If now $s_k = 0$, which has probability 1/2, independently of the condition on H, then Gm will assign message(m) := temp(m) = V at the end of the k-th invocation of Decision.

The other possibility is that for every V,

(1) $|\{j : Gj \in H,\ V = Gj\text{'s k-th value of message}(j)\}| < n - 4t$.

Let again Gm be any proper process. Among the $n - t$ responses σ_j(message(j),k) it collects there are at least $n - 3t$ from $Gj \in H$. Thus at most 2t responses from outside of H. Even if all these responses have the form $\sigma_\ell(V,k)$ for some common V, and this V is the most popular version, (1) will imply that count(m) $< n - 4t + 2t = n - 2t$ at the end of the k-th invocation of Polling.

Thus count(m) $< n - 2t$ holds for every proper Gm. If now $s_k = 1$ then at the end of the k-th invocation of Decision all the proper processes will have message(m) = "system faulty". This again will happen with probability 1/2.

Once agreement is reached it persists, so the probability for not reaching agreement in R rounds is 2^{-R}.

6. BOUNDED EXPECTED TIME, ERRORLESS SOLUTION

We turn now to a BAP which ensures a Byzantine agreement without error, and reaches agreement within a small expected number of rounds, independently of n and t, as long as $t < n/10$.

The key observation is that if a process Gi finds, during the k-th iteration of Decision, that $s_k = 0$, and $n-2t \leq$ count(i) and $s_k = 0$, then necessarily every other proper Gj will have $n/2 < n-4t \leq$ count(j) during its k-th round of Decision. Hence according to the If statement in Decision, every proper Gj will assign message(j) := temp(j) at the end of this invocation of

Decision. By the proof of Theorem 1, all values temp(j) for proper Gj will then be the same. Thus if the above condition holds for some proper Gi, it constitutes proof for the fact that all proper Gj will attain the same value for message(j) at the end of their k-th invocation of Decision.

To utilize this fact, we modify Decision. We introduce an integer variable k(i) which is local to Gi but global within Pi.

```
Procedure Decisionproof; {for Gi}

  begin
    s := s_k(i); {s_k(i) is current shared
                  secret}
    If (s = 0 and n/2 ≤ count(i)) or (s = 1)
       and n - 2t ≤ count(i)) then
       message(i) := temp(i) {= V}
    else
       message(i) := "system faulty";
    If (s = 0 and n - 2t ≤ count(i)) then
       send σi ("agreement reached on V")
         to all Gj;
    k(i) := k(i) + 1 {update the index of the
         secret to be shared}
  end; {Decisionproof}
```

Since there will be no fixed number of phases before stopping, we shall need a Boolean-valued variable inround(i) to provide Pi with a halt-signal. The BAP (actually Pi) will now read

```
Procedure ETBAP; {expected time protocol}

  begin
    inround(i) := true;
    k(i) := 1;
    message(i) := Mi  {the initial message
                       for Gi}
    repeat
      begin
        Polling;
        Lottery;
        Decisionproof {increases k(i) by 1
                        each time}
      end
  end; {ETBAP}
```

Lemma 2. Within an expected number of four rounds of iterations of the loop of ETBAP, every proper Gj will send σ_j("agreement reached" on V) for some V.

At the time that any proper Gj sends σ_j ("agreement reached on V"), it is already determined that all the proper Gi's will in fact reach Byzantine Agreement on the value V.

Proof. The procedure Decisionproof is an extension of Decision. Hence it follows, by the argument in the proof of Theorem 1, that within an expected number of two iterations of the loop, the proper processes will have reached Byzantine Agreement. This means that for some V, which if the system is proper is its original common message, for all proper Gi we have message(i) = V.

Within an expected number of two additional iterations of the loop, the value $s_k = 0$ for the shared random secret will appear. At that time (on their asynchronous phase-clocks) all proper Gj's will send σ_j ("agreement reached on V") during Decisionproof.

As ETBAP stands it does not include a stopping rule. It will not do to stipulate than any Gj which has sent σ_j (agreement reached") also assigns inround(j) := false and stops. For the above state may be reached by some proper Gj before others, and the late proper processes will need cooperation of the early ones in order to reach that state.

We therefore introduce an additional procedure Closefinish which runs simultaneously with ETBAP and provides the stopping rule.

```
Procedure Closefinish; {for Gi}

  begin
    repeat
      If received new σj("agreement reached
         on V") {Gi's own communication of
         this form is included.  V may
         differ for faulty  Gi}
      then send σj("agreement reached V")
         to all
    until t + 1 communications with the same
      V are counted;
    message(i) := V;
    inround(i) := false
  end; {Closefinish}
```

Now, the value inround(i) = false is construed as an interrupt which will cause immediate termination of, and exit from, the repeat statement of ETBAP.

Note that if a proper process has received t + 1 communications σ_j("agreement reached on V") then at least one of these is from a proper Gj. That proper Gj actually had a proof that agreement will be reached on the value V. Thus Gi correctly stops. The complete proof of correctness is omitted here.

7. CONCLUDING REMARKS

We have fully treated in this paper the asynchronous case of the BGP. In the synchronized case it is assumed that the processes Gi have phase-clocks p(i), and that these phases have the same start and end-points for all processes; this is the partially synchronized case. If we further assume that the processes have the same clock-reading at any given time, then we have the fully synchronized case.

The availability of synchronization enables us to simplify our protocol. It now suffices to require $t < n/4$. A new issue that arises in the synchronized case is that of coordinated agreement, namely we may want the proper processes to reach final decision about the common message at the same time. The author has an algorithm for enhancing any BA protocol to one that produces coordinated agreement. H. Gaifman produces coordination by use of randomization along the lines of our paper.

Other formulations of the BGP assume a sender G (who could also be anyone of the Gj) who sends each Gi a message which becomes its initial message Mi. The case without sender is sometimes called the Byzantine Consensus Problem. The case where there is a sender is readily reduced to the one treated here. All that is additionally required is a Wakeup protocol where any proper process, upon first receiving the sender's message, broadcasts it.

As written here, our solutions require a total of $O(n^2)$ messages. But this can be readily reduced to $O(nt)$.

Finally, the solution extends to cover the situation that the system has to repeatedly reach agreement on new sender's messages. Under appropriate assumptions the solution is resilient. Transient failures of processes in one agreement round, do not affect subsequent rounds.

Details of the above work will be given in the full version of the paper.

BIBLIOGRAPHY

[DS81] Dolev, D., Strong, H.R., Polynomial algorithms for multiple processor agreement, IBM Research, San Jose, 1982.

[D82] Dolev, D., The Byzantine Generals strike again, J. of Algorithms, vol. 3 (1982).

[LF82] Lynch, N., Fischer, M., A lower bound for the time to assure interactive consistency, Inf. Proc. Letters, Vol. 4 (1982), pp. 183-186.

[LSP80] Lamport, L., Shostak, R., and Pease, M., The Byzantine Generals problem, Tech. Report 54, Comp. Sc. Lab., SRI Inter., 1980.

[BO83] Ben-Or, M., Another advantage of free choice: Completely asynchronous agreement protocols, Abstract.

[Ra76] Rabin, M.O., Probabilistic algorithms, Algorithms and Complexity, J.F. Traub, Editor, Academic Press (1976), pp. 21-39.

[FLP82] Fischer, M., Lynch, N., and Paterson, M., Impossibility of distributed consensus with one faulty process, MIT/LCS/TR-282.

[RSA78] Rivest, R., Shamir, A., and Adleman, L., A method for obtaining digital signatures and public-key cryptosystems, CACM, Vol. 21 (1978), pp. 120-126.

[Sh79] Shamir, A., How to share a secret, CACM, Vol. 22 (1979), pp. 612-613.

[DH76] Diffie, W., Hellman, M., New directions in cryptography, IEEE Trans. Inf. Theory, IT-22 (1976), pp. 644-655.