Randomized Byzantine Agreement

Kenneth J. Perry

TR 84-595
March 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

Randomized Byzantine Agreement

Kenneth J. Perry

Department of Computer Science
Cornell University
Ithaca, New York 14853

ABSTRACT

A randomized model of distributed computation was presented in [Rabin83]. This model admits a solution to the Byzantine Agreement Problem for systems of n asynchronous processes where no more than t are faulty. The algorithm described in [Rabin83] produces agreement in an expected number of rounds which is a small constant independent of n and t. Using the same model, we present an algorithm of similar complexity which is able to tolerate a greater proportion of malicious processes. The algorithm is also applicable, with minor changes, to systems of synchronous processes.

1. Introduction

Consider a collection of n processes, each possessing a previously initialized local variable *message*. Desired is an *agreement protocol* whose execution results in the same value being assigned to the *message* variables of all processes. Some processes may not obey the protocol for its entire duration. In fact, these processes may actively attempt to hinder agreement. We call any process that deviates from the protocol *malicious*. Those processes that strictly adhere to the protocol for its entire execution are deemed *proper*. Because we can not constrain the behavior of malicious processes, we require only that all proper processes agree on their *message* values upon termination.

A *Byzantine Agreement* protocol is an agreement protocol with the additional constraint that if the protocol commences with the same value V in the *message* variables of all proper processes, then V is the value agreed upon on termination. The problem of devising Byzantine Agreement protocols has been extensively studied [PSL80], [LSP82], [DS82], [LFF82], [DDS83].

In this paper, we present a Byzantine Agreement protocol based on a model of computation first proposed in [Rabin83]. The protocol proceeds as a repetition of *phases*. (A phase is defined to be the interval in which the processes exchange *message* values). If we iterate for a fixed number of phases, then there is a non-zero probability of disagreement upon termination. However, we can make this probability arbitrarily small. We may choose instead to iterate indefinitely so that the protocol terminates only when agreement has been reached. Termination occurs with probability 1. Specifically, we shall prove correct a protocol for n processes where no more than t are malicious. When the collection of processes is asynchronous, n must be at least $6t + 1$. With minor modifications, the same protocol is correct for a system of $n \geq 3t + 1$ synchronous processes.

[Rabin83] allows the *message* values to be drawn from an arbitrary universe. The approach adopted in this paper is to first present a solution in which the *message* values are single bits and then give a method for converting such a solution into one where the values may be drawn from an arbitrary universe. Our technique of converting a binary Byzantine Agreement protocol to one that works for arbitrary values is independent of the model of computation used in this paper.

1.1. Notational conventions

The following notational conventions are used throughout this paper in order to eliminate ambiguity when it arises:

(1) All variables local to process P_i are subscripted with i.

(2) A variable is superscripted with R to indicate the value assigned to it in phase R. Superscript 0 is used to denote an initial value, that is, one set prior to invocation of the protocol.

Thus $message_i^R$ is the value assigned to process P_i's *message* variable in phase R.

1.2. Shared secrets

The distinguishing characteristic of the model of [Rabin83] is its reliance on shared, randomly chosen values. Using Shamir's technique for sharing secrets [Shamir79], any set of $t + 1$ or more processes may cooperate to uncover a bit, randomly chosen in advance by a "dealer". We briefly describe the technique.

Embedded as data into the protocol executed by process P_i are "pieces" of N secrets. Let p_i^m denote the piece of the m^{th} secret embedded in process P_i. When combined with any t other pieces of the m^{th} secret, a unique bit is revealed to P_i. Let **ComputeSecret** be the function that maps $t + 1$ pieces of a secret into the secret itself. Before the Byzantine Agreement protocol is first run, a "dealer" choses N bits $b^1, \ldots, b^N$ at random. Each bit b^m is used to derive n pieces $p_1^m, \ldots, p_n^m$ with piece p_i^m being distributed to process P_i. By cooperating with t other processes, any process is able to discover b^m. Therefore this method allows all processes to agree on a value.

To insure that a malicious process does not supply an invalid piece of a secret, the "dealer" signs each, thereby rendering it unforgeable. Let

$$s_i^m = \mathbf{DealersSignature}(<p_i^m,\ m>)$$

represent the signed piece of the m^{th} secret given to process P_i. (**Note**: The second component allows a process to verify that a piece received from another process is, in fact, part of the m^{th} secret). Signatures may be obtained by using a public-key cryptosystem such as that of [RSA78].

1.3. The message system

For ease of explanation, a message m is considered to be a tuple with the following elements:

- *m.type* $\in \{MessageValues,\ Secrets,\ InitialValues,\ SecondValues,\ AgreementValues\}$ describing the message's purpose.
- *m.phase* indicating the phase in which the message was sent.
- *m.value*, the message's value.

We make the following assumption, in common with most solutions to the Byzantine Agreement problem:

Assumption 1.1

The receiver of message m may determine the identity of the process which sent m.

This is trivially true when the network is completely connected. We use *m.sender* to denote the process that sent message m.

[Rabin83] avoids this assumption by requiring messages to be signed. Hence, his solution is an authenticated one. We find this unnecessary and prefer the above assumption instead. *This is the only aspect in which our model and that of* [Rabin83] *differ.* [Rabin83] requires the following assumption which we adopt as well:

Assumption 1.2

All messages sent between proper processes are eventually received.

1.4. Exchange protocol

So that processes may exchange *message* values, we describe a family of procedures **Exchange**^{T} which are generalizations of the **Polling** procedure described in [Rabin83].

Definition: Message m is a *Type T message valid for R* if

$m.type = T$ **and** $m.phase = R$

Process P_i invokes **Exchange**^{T} in phase R to establish the following post-condition:

$message_i^{R-1}$ has been sent to all processes (including itself. Properly speaking, the tuple $<T,\ R,\ message_i^{R-1}>$ is sent)

and

For each value $V \in A_i^R$:

$$count_i^R(V) = \left| \{ m \in A_i^R \mid m.value = V \} \right|$$

where A_i^R is a set containing the Type T messages valid for R received by P_i from distinct processes.

When we are dealing with systems of asynchronous processes, A_i^R will contain the first $(n-t)$ Type T messages valid for R received by P_i. For synchronous systems, A_i^R contains all n Type T messages sent by the end of phase R. (A malicious process that sends no value to P_i during the phase is considered to have sent the ϕ value). **Exchange**^{MessageValues} is the equivalent of procedure **Polling** of [Rabin83]. Note however that **Exchange** does not require signatures.

1.5. Lottery

A process joins in the lottery of phase R so that the R^{th} secret bit may be revealed. Rabin describes a procedure **Lottery** which is executed by process P_i in phase R to establish the post-condition:

s_i^R has been sent to all processes (including itself) **and** $secret_i^R = b^R$

where b^R is the R^{th} secret chosen by the "dealer". (Properly speaking, the tuple $<Secrets, R, s_i^R>$ is sent).

We outline a means by which the second conjunct may be implemented.

Definition: Message m is a *Secret message valid for R* if

$m.type = Secret$ **and** $m.phase = R$ **and**

$m.value$ is of the form **DealersSignature**($<p_j^R, R>$)

b^R may be obtained by accumulating Secret messages valid for R from t other processes and applying **ComputeSecret**.

2. The algorithm

The Byzantine Agreement protocol is presented in Figure 2.1 which follows. It proceeds as a repetition of phases. When *LastPhase* is a finite constant M, the protocol terminates in exactly M rounds with a probability of agreement equal to $1-2^{-M}$. If definite agreement is desired, consider *LastPhase* to be a variable, initially equal to ∞, which is computed on each phase by the **CloseFinishPhase** procedure described in the appendix.

In the case of asynchronous processes, consider $n \geq 6t + 1$ and $C = \left\lfloor \frac{n + 2t}{2} \right\rfloor$. For systems of $n \geq 3t + 1$ synchronous processes, let $C = \left\lfloor \frac{n + t}{2} \right\rfloor$.

We again emphasize that this algorithm works for the case where *message* values are drawn from the set $\{0, 1\}$. Subsequently we will present a method for using this algorithm to reach agreement on values drawn from an arbitrary universe.

3. Correctness

The correctness of **ByzantineAgreement** is proved in this section. We begin with some definitions that will facilitate elaboration of the proof.

Definition: The *configuration at time R* of a system of processes is the collection of values in the *message* variables of **proper** processes at the start of phase R. (i.e. the $(n-t)$ values assigned to the *message* variables of proper processes during phase $R-1$).

A *proper configuration* is a configuration in which all proper processes have the same *message* value.

```
proc Decision
  begin
    if count_i^R(0) > count_i^R(1)
      then majority := 0
      else majority := 1;

    if count_i^R(majority) > C
      then message_i^R := majority
      else message_i^R := secret_i^R;
  end

proc Phase^R
  begin
    Exchange^MessageValues;
    Lottery;
    Decision
  end

proc ByzantineAgreement
  begin
    R := 0;
    do while (R ≠ LastPhase)
      begin
        R := R + 1;
        Phase^R;
      end
  end
```

Figure 2.1

Definition: A configuration at time R is *biased toward* V if it contains in excess of $(C-t)$ *message* values equal to V. V is called the *bias* of the configuration.

Note that a configuration of asynchronous processes need not be biased. However, when a bias exists, it is unique.

Definition: We say that proper process P_i is *forced to* V during phase R if $count_i^R(V) > C$.

We begin the proof by introducing several useful lemmas. First, we prove that, in any phase, a proper process may be forced only to the bias (if it exists) of the configuration at the phase's start.

Lemma 3.1

A process can be forced to V during phase R only if V is the bias of the configuration at time R.

Proof

Suppose process P_i is forced to V during phase R. Then P_i received value V from at least $(C+1)$ processes during phase R. Because there are at most t malicious processes, the configuration contains at least $(C+1-t)$ processes with *message* value V. Thus, V is the bias. □

The next lemma shows that the shared secrets are not revealed prematurely.

Lemma 3.2

The m^{th} shared secret bit b^m can not be discovered by any process (whether malicious or not) until the (potential) bias of the configuration at time m is fixed.

Proof

The Lemma follows trivially when the processes are synchronous. Consider the asynchronous case. For the m^{th} secret to become known, at least one proper process P_i must reveal its piece. This it does only **after** terminating the $\mathbf{Exchange}^{MessageValues}$ step of phase m. Included in the values received by P_i during this step is a set S of at least $(n-2t)$ messages sent by proper processes. Let V be the majority value in S. Even if the (at most) t proper processes not in S have value $\overline{V}$, the number of proper processes beginning phase m with *message* value $\overline{V}$ is insufficient to bias the configuration at time m toward $\overline{V}$ (since $\left\lfloor \frac{n-2t}{2} \right\rfloor + t \leq C-t$). Hence, V is the only potential bias of the configuration at time m. □

The next theorem is a key one in the proof of correctness of the protocol. It shows that there is some probability of reaching agreement during any phase. Therefore, in spite of the best efforts of the malicious processes, agreement can not be indefinitely prevented.

Theorem 3.3

If the configuration at time R is not proper, then the configuration at time $R+1$ is not proper with probability $\frac{1}{2}$.

Proof

Suppose the configuration at time R is not proper. Consider the time at which the first proper process P_i terminates the **Exchange**^{MessageValues} step of phase R. At this moment, the (potential) bias of the configuration at time R assumes a fixed value V. (That is, if a bias exists, it must equal V. Whether or not V becomes the actual bias depends on the behavior of the proper processes which have not reached phase m by this time). Only now is the R^{th} secret bit b^R revealed to the malicious processes. (These last two claims follow directly from Lemma 3.2). Consider any other proper process P_j. According to Lemma 3.1, P_i and P_j may only be forced to V. Our method of sharing secrets insures that P_i and P_j agree on $secret^R$. Hence, if procedure **Decision** causes P_i and P_j to take on differing $message^R$ values, it must be the case that one process was forced to V and the other was not (in which case it uses $secret^R$ as its $message^R$ value). But since $secret^R = b^R$ was randomly chosen, $V \neq secret^R$ with probability $\frac{1}{2}$. Therefore two processes assume different $message^R$ with probability $\frac{1}{2}$. □

We now show that agreement is persistent. That is, once all proper processes have reached agreement, they will not deviate from it.

Theorem 3.4

Consider a system of n asynchronous processes with at most t faulty. Let $n \geq 6t + 1$. If the configuration at time R is proper with V as the only value, then the configuration at time $R + 1$ is also proper with V as the sole value.

Proof

Assume that the configuration is proper at time R and that for each proper process P_i

$$message_i^{R-1} = V$$

By construction of procedure **Exchange**^{MessageValues}, a proper process observes $n - t$ values, of which at least $n - 2t$ are from proper processes. So for every proper process P_i

$$count_i^R(V) \geq n - 2t$$

Because $n \geq 6t + 1$, $n - 2t > C$. Thus, proper processes are forced to V in phase R. Therefore, for all proper P_i

$$message_i^R = V$$

□

Lemma 3.5

Agreement is persistent. That is, a proper configuration at time R may never become improper.

Proof

By induction on the number of phases executed after R. □

The resulting corollary is also a key step in the protocol's correctness proof. It proves that if all proper processes start the protocol with the same value V in their *message* variables, then V is the *message* value upon termination. This is a requirement of any Byzantine Agreement protocol.

Corollary 3.6

If the initial configuration is proper with V as the only value, then it remains proper with V as the only value.

Correctness of the protocol has now been proved. We've shown that, with great probability, an improper initial configuration will not remain so indefinitely. (By Theorem 3.3, the probability that an initially improper configuration is still not proper after R phases is $1-2^{-R}$. Thus, the probability of agreement tends toward 1 as R increases). It has also been shown (Lemma 3.5) that once a configuration has been made proper it will never return to an improper state. Finally, Corollary 3.6 demonstrates that when the initial configuration is proper with V as its only value, then V is the *message* ~~value ultimately~~ decided upon by all proper processes. Therefore, we have proven the following

Theorem 3.7

ByzantineAgreement is correct.

Proof

By Theorem 3.3, Lemma 3.5, and Corollary 3.6. □

4. An algorithm for a system of $n \geq 3t + 1$ synchronous processes

In this section we prove the correctness of the protocol for a system of $n \geq 3t + 1$ synchronous processes.

The synchronous protocol differs from the asynchronous version in the number of messages received by **Exchange**^{T} and in the value of C. In an asynchronous system, requiring proper processes to wait for more than $(n-t)$ messages gives the malicious processes a means of foiling agreement. That is, by never sending messages, they cause proper processes to wait indefinitely. However, there is no such worry in synchronous systems. We simply ignore any process that has not communicated by the phase's end. Synchrony means that, during each phase, a proper process receives the *message* values of **all** proper processes. This enables us to tolerate more malicious processes than in an asynchronous system.

Correctness of the protocol for a system of $n \geq 3t+1$ synchronous processes is proved in a manner almost identical to the one used for the asynchronous system. Only the proof of Theorem 3.4 needs to be changed to reflect the fact that $n \geq 3t+1$.

Theorem 4.1

Consider a system of n synchronous processes with no more than t faulty. Let $n \geq 3t+1$. If the configuration at time R is proper with V as the only value, then the configuration at time $R+1$ is proper with V as the sole value.

Proof

By construction of the synchronous version of procedure **Exchange**^{MessageValues}, all proper processes receive the $(n-t) > C$ $\left(=\left\lfloor\frac{n+t}{2}\right\rfloor\right)$ values V sent by the proper processes during phase R. Hence, all proper processes are forced to V in phase R. Therefore, for all proper P_i

$$message_i^R = V$$

□

With this change, the correctness of the protocol for synchronous systems follows from the proof of the asynchronous protocol.

Theorem 4.2

Consider a system of $n \geq 3t+1$ synchronous processes with at most t faulty. Then **ByzantineAgreement** is correct.

5. Reaching agreement on arbitrary values

In this section, we present a method whereby an algorithm for Byzantine Agreement on a binary valued message may be converted to one that reaches agreement on a value from an arbitrary universe. The method is independent of the model of [Rabin83] or that of this paper. Our solution requires that additional phases *Cnvt*1 and *Cnvt*2 (Figure 5.1) be interposed between phases 0 and 1. For these two phases, consider $C = \left\lfloor \frac{n+t}{2} \right\rfloor$.

5.1. Correctness of General Byzantine Agreement

The correctness of **GeneralByzantineAgreement** is now proved. We begin by observing that the bias of a configuration is still well defined even if the *message* values are not single bits. It is clear that a configuration has at most one bias and that this is the only value to which a proper process may be forced during the conversion phases. (Proof of the preceding claim is nearly identical to the one used in Lemma 3.1). The intuition behind our method is that the initial bias becomes a potential final *message* value only if it is known by all processes. This is formalized in the theorem which follows. It will be useful to first state a lemma which we prove correct both for asynchronous ($n \geq 6t + 1$) and synchronous ($n \geq 3t + 1$) systems of processes.

Lemma 5.1

Consider any pair of proper processes P_i and P_j. If during phase R, P_i receives in excess of $\left\lfloor \frac{n+t}{2} \right\rfloor$ Type T messages valid for R with value V, then P_j receives at least $t + 1$ of these messages.

Proof

Suppose P_i receives in excess of $\left\lfloor \frac{n+t}{2} \right\rfloor$ Type T messages valid for R with value V. At least $\left\lceil \frac{n+t}{2} \right\rceil - t$ of these messages are from proper processes which send the same message to all.

Suppose the system consists of $n \geq 6t + 1$ asynchronous processes. In the asynchronous version of **Exchange**^{T}, messages from t processes are ignored by a proper process during any phase.

```
proc ConvertToBinary^Cnvt1
  begin
    message_i^Cnvt1 := message_i^0;

    Exchange^InitialValues;
    Plurality := the value V with the greatest count_i^Cnvt1(V);

    if count_i^Cnvt1(Plurality) > ⌊(n + t)/2⌋
      then message_i^Cnvt1 := Plurality
      else message_i^Cnvt1 := '?';
  end

proc ConvertToBinary^Cnvt2
  begin
    Exchange^SecondValues;
    if ∃ V ≠ '?': count_i^Cnvt2(V) > t
      then OneValue := V
      else OneValue := φ;

    if count_i^Cnvt2(OneValue) > ⌊(n + t)/2⌋
      then message_i^Cnvt2 := 1
      else message_i^Cnvt2 := 0;
  end


proc GeneralByzantineAgreement
  begin
    ConvertToBinary^Cnvt1;
    ConvertToBinary^Cnvt2;

    ByzantineAgreement;

    if message_i^R = 1
      then message_i^General := OneValue
      else message_i^General := 'DefaultValue'
  end
```

Figure 5.1

Hence, P_j receives at least $\left\lceil \frac{n+t}{2} \right\rceil - 2t > t$ Type T messages valid for R with value V. Suppose instead that we are dealing with a system of $n \geq 3t + 1$ synchronous processes. The synchronous version of **Exchange**^{T} insures that all messages sent by proper processes are received by P_j. Since $\left\lceil \frac{n+t}{2} \right\rceil - t > t$, our claim is proved. □

Theorem 5.2

Let V be the bias of the initial configuration, if it exists. If there is a proper process P_i such that $message_i^{Cnvt2} = 1$, then for all proper P_j

$$OneValue_j^{Cnvt2} = V$$

Proof

Suppose $message_i^{Cnvt2} = 1$. Then it must be the case that P_i received in excess of $\left\lfloor \frac{n+t}{2} \right\rfloor$ Type *SecondValues* messages with value $OneValue_i$ during $\mathbf{Exchange}^{SecondValues}$. By Lemma 5.1, at least $t+1$ of these messages are also received by any other proper P_j. Therefore

$$OneValue_j = OneValue_i$$

by construction of $\mathbf{ConvertToBinary}^{Cnvt2}$.

That $OneValue_i = V$ is seen as follows. At least one proper process must send V during $\mathbf{Exchange}^{Cnvt2}$. $\mathbf{ConvertToBinary}^{Cnvt1}$ is designed such that a proper process P_k assigns a non-'?' value $\tilde{V}$ to its *message* variable only if it is forced to $\tilde{V}$ in phase *Cnvt* 1. But a process may be forced only to the bias V. Hence V is the only non-'?' value sent by a proper process in $\mathbf{Exchange}^{SecondValues}$. □

Correctness of **GeneralByzantineAgreement** is proved in the next two theorems.

Theorem 5.3

If the initial configuration is proper with V as its only value, then at the end of **General-ByzantineAgreement**, for each proper process P_i

$$\bar{message}_i^{General} = V$$

Proof

Because the initial configuration is proper, all proper processes are forced to V during phases *Cnvt* 1 and *Cnvt* 2. (This is true both for systems of $n \geq 6t+1$ asynchronous processes and $n \geq 3t+1$ synchronous processes). Thus all proper processes begin **ByzantineAgreement** with $message_i = 1$ and $OneValue_i = V$. By Theorem 3.7 (resp. Theorem 4.2 for synchronous systems) **ByzantineAgreement** terminates with all proper processes having 1 as the *message* value. The final conditional statement of **GeneralByzantineAgreement** results in

$$message_i^{General} = V$$

for all proper P_i. □

Theorem 5.4

If the initial configuration is not proper, **GeneralByzantineAgreement** reaches agreement.

Proof

By Theorem 3.7 (resp. Theorem 4.2), there exists a value V^{byz} such that, at the end of **ByzantineAgreement**,

$$message_i = V^{byz}$$

for all proper P_i.

Suppose $V^{byz} = 0$. Then the final conditional of **GeneralByzantineAgreement** results in

$$message_i^{General} = \text{'}DefaultValue\text{'}$$

for all proper processes P_i.

The only other possibility is that $V^{byz} = 1$. In this case, at least one proper process P_i began **ByzantineAgreement** with $message_i = 1$ (Otherwise the initial configuration for **ByzantineAgreement** would be proper with 0 as its only value. By the correctness of **ByzantineAgreement**, $V^{byz} = 0$, a contradiction). Therefore, by Theorem 5.2, for all proper P_j

$$OneValue_j = V$$

Thus, the final conditional of **GeneralByzantineAgreement** results in

$$message_i = V$$

for all proper processes P_i. □

We have therefore demonstrated that

Theorem 5.5

GeneralByzantineAgreement is correct.

Proof

By Theorems 5.3 and 5.4. □

6. Complexity

The complexity of the algorithm is addressed in this section. By Theorem 3.3 we conclude that in an expected two phases an improper initial configuration becomes proper. Additional phases thereafter leave a proper configuration unchanged. Hence, we will have agreement but iterate until our termination condition becomes true.

A necessary and sufficient condition for terminating the algorithm is presented in the appendix. In that section we prove that the termination condition becomes true by an expected one additional phase after agreement has been reached. Therefore the algorithm for reaching agreement on a single bit has an expected running time of three phases.

If agreement on values drawn from an arbitrary universe is desired, the **ConvertToBinary**^{Cnvt1} and **ConvertToBinary**^{Cnvt2} may be executed at the cost of two additional phases.

7. Acknowledgements

The author would like to express his appreciation to Sam Toueg for the insights and encouragement provided during the course of this work.

8. Appendix: Detecting Agreement

[Rabin83] describes a procedure **CloseFinish** which runs in parallel with the agreement protocol and causes its termination. We can adapt this procedure to accomplish the same ends (i.e. determining which phase is last) at the cost of **requiring** signatures. In this section we show that termination may be detected without a signature capability. The technique employed is reminiscent of one used in [DS82] and [LFF82].

Definition: Suppose proper process P_i is forced to V in phase R and that the R^{th} shared secret $b^R = V$. Then we say that P_i has a *proof of agreement on* V.

Because agreement is persistent, proof of agreement on V in phase R implies that for phases $\hat{R} \geq R$, $message_j^{\hat{R}} = V$ for all proper P_j. Having proof of agreement is alone insufficient to enable a process to end participation in the protocol. It is safe to terminate only when **all** proper processes obtain proofs. We facilitate this by allowing "indirect" proofs of agreement.

Definition: If proper process P_i receives at least $t + 1$ messages, each purporting that its sender has a proof of agreement on V, then we say that P_i has an *indirect proof of agreement on V*.

In any set of $t + 1$ or more processes there is at least one proper process that really has a proof. Hence, an indirect proof is as valid as a direct one. With **CloseFinish** we establish the condition that each proper process obtains a proof of agreement on the same value.

By Lemma 5.1 we know that when any process receives $\left\lceil \frac{n+t}{2} \right\rceil$ or more claims of proof, then all processes simultaneously receive enough claims to achieve indirect proof. It will be shown that if any phase begins with each proper process having a proof, then all receive $\left\lceil \frac{n+t}{2} \right\rceil$ claims of proof in that phase. Hence, receiving $\left\lceil \frac{n+t}{2} \right\rceil$ claims of a proof is a sufficient condition for termination. We formalize this method in the algorithm of Figure 8.1.

CloseFinish is run by an additional parallel process (local to each process participating in the agreement protocol) which executes a phase of termination detection **after** each phase of the agreement algorithm. Shared variable *LastPhase* signals termination. Rather than considering termination detection as a separate process, it is notationally more convenient to embed a call to **Phase** (of the agreement algorithm) within each phase of termination detection. Thus, one may view **CloseFinish** as the **agreement protocol itself** rather than a parallel process.

Information for each phase of **CloseFinish** is exchanged in a manner similar to the **Exchange** protocol previously described. Process P_i invokes $\mathbf{Exchange}^{AgreementValues}$ in phase R to establish the postcondition:

> $AgreementValue_i^{R-1}$ has been sent to all processes (including itself. Properly speaking, the tuple $<AgreementValues, R, AgreementValue_i^{R-1}>$ is sent)
>
> **and**
>
> For each value $V \in T_i^R$:
>
> $$AgreedCount_i^R(V) = \left|\{m \in T_i^R \mid m.value = V\}\right|$$
>
> <u>where</u> T_i^R is a set containing the Type *AgreementValues* messages valid for R received by

P_i from distinct processes.

The size of T_i^R is defined to be $(n-t)$ for systems of asynchronous processes and n for systems of synchronous processes.

---

proc CloseFinishPhase^R
  begin
    { Is there a direct proof of agreement ? }
      if $AgreementValue_i = \phi$ **cand** $count_i^R(majority_i^R) > \left\lfloor \frac{n+t}{2} \right\rfloor$ **and** $secret_i^R = majority_i^R$
        then
          $AgreementValue_i := majority_i^R$;

    Exchange^{AgreementValues};

    { Is there an indirect proof of agreement ? }
      if $AgreementValue_i = \phi$ **and** $\exists\, V \neq \phi : AgreedCount_i^R(V) > t$
        then
          $AgreementValue_i := V$;

    if $AgreementValue_i \neq \phi$ **and** $AgreedCount_i^R(AgreementValue_i) > \left\lfloor \frac{n+t}{2} \right\rfloor$
      then $LastPhase := R$
  end

proc CloseFinish
  begin
    $R := 0$; $LastPhase := \infty$;

    $AgreementValue_i := \phi$;
    $AgreedCount(\phi) := 0$;

    do while $(R \neq LastPhase)$
      begin
        $R := R + 1$;
        {**Phase**^{R}; }
        CloseFinishPhase^R
      end;

    send($<AgreementValues,\ LastPhase + 1,\ AgreementValue_i>$) to all processes;

   $message_i^R := AgreementValue_i$

  end

Figure 8.1

8.1. Correctness

Consider a system of $n \geq 6t + 1$ asynchronous processes such that no more than t are faulty. (For the synchronous analogues of these proofs, let $n \geq 3t + 1$). The next two theorems show that a proper process may set *AgreementValue* only to a value which is common to all processes.

Theorem 8.1

Let R' be the earliest phase in which any proper process P_i assigns a non-ϕ value V to its *AgreementValue* variable. Then for all phases $\hat{R} > R'$, the configuration at time $\hat{R}$ is proper with V as its only value.

Proof

By the minimality of R', P_i must have a direct proof of agreement on V. That is

$$secret_i^{R'} = majority_i^{R'} = V$$

By construction of **Phase**$^{R'}$ of **ByzantineAgreement**

$$message_j^{R'} = b^{R'} = V$$

for all proper processes P_j, where $b^{R'}$ is the R'^{th} shared secret. Therefore the configuration becomes proper during phase R' and remains so thereafter (Lemma 3.5). □

The next theorem shows that a proper process never obtains a proof of agreement on an incorrect value.

Theorem 8.2

Let R' be the earliest phase in which any proper process P_i sets its *AgreementValue* variable to a non-ϕ value V. If a proper process P_j ever assigns a value $\tilde{V}$ to its *AgreementValue* variable, then $\tilde{V} = V$.

Proof

By construction of **CloseFinishPhase**, a proper process never changes its *AgreementValue* variable once it has been set. Therefore, once set to the correct value, it will never change.

Suppose P_j assigns $\tilde{V}$ to $AgreementValue_j$ during phase R'. By the argument of Theorem 8.1, $\tilde{V} = V$.

On the other hand, suppose P_j makes the assignment in a later phase $\hat{R}$. If the assignment is performed because P_j has a direct proof of $\tilde{V}$, then $\tilde{V} = V$ by Theorems 3.1 and 8.1. The only other condition under which assignment to $AgreementValue_j$ is made is when

$AgreedCount(\tilde{V}) > t$. But then at least one proper process sent $\tilde{V}$ to P_j. By backward induction on phase numbers we see once more that $\tilde{V} = V$. □

We now show that all proper processes obtain proofs of agreement on V by the earliest phase in which any process terminates.

Theorem 8.3

Let $R' \neq \infty$ be the first phase in which any proper process P_i causes $LastPhase_i$ to assume a non-∞ value. There is a value $V \neq \phi$ such that $AgreementValue_j^{R'} = V$ for all proper P_j.

Proof

Suppose P_i makes $LastPhase_i$ non-∞ during phase R'. Then there is a non-ϕ value V such that $AgreedCount_i^{R'}(V) > \left\lfloor \frac{n+t}{2} \right\rfloor$. By Lemma 5.1, each proper P_j receives at least $t+1$ of these messages in phase R'. Therefore, proper process P_k assigns $AgreementValue_k^{R'} = V$ if it has not done so previously. □

Next, we show that once all proper processes have proofs of agreement, termination occurs.

Theorem 8.4

If there is a phase $R \neq \infty$ and value $V \neq \phi$ such that the $\mathbf{Exchange}^{AgreementValues}$ step of phase R begins with

$$AgreementValue_i = V$$

for each proper P_i, then $LastPhase_i^R \neq \infty$ for all proper P_i.

Proof

Suppose that at the start of the $\mathbf{Exchange}^{AgreementValues}$ step of phase R, each proper process has the same non-ϕ value V in its $AgreementValue$ variable. During $\mathbf{Exchange}^{AgreementValues}$, at least $(n-t)$ processes send V to all processes. Each proper process receives at least $(n-2t)$ of these messages ($(n-t)$ for synchronous processes). But, for $n \geq 6t+1$ (resp. $n \geq 3t+1$ synchronous)

$$n - 2t \text{ (resp. } n - t \text{ synchronous)} > \left\lfloor \frac{n+t}{2} \right\rfloor$$

Therefore all proper processes assign R to their $LastPhase$ variable. □

Combining these results validates the termination condition:

Theorem 8.5

If any proper process terminates with proof of agreement on $V \neq \phi$, then all proper processes terminate with proofs of agreement on V within one additional phase.

Proof

By Theorems 8.2, 8.3 and 8.4. □

Finally we demonstrate that if agreement is reached, it will eventually be detected.

Theorem 8.6

If the configuration is proper at time R, then in an expected one additional round **ByzantineAgreement** terminates.

Proof

Suppose the configuration is proper at time R with V as its only value. Then all proper processes are forced to V in phase R and all later phases as well. In two consecutive phases R and $R + 1$, the expected number of times that the random secret equals V is once. Thus, by expected round $R + 1$ each proper processes P_i sets

$$AgreementValue_i = V$$

before running $\textbf{Exchange}^{AgreementValues}$. By Theorem 8.4, all processes terminate as soon as this occurs. □

Therefore, we have shown

Theorem 8.7

If agreement is reached, then all processes eventually terminate with the agreed-upon value. No process may terminate unless all terminate with the same correct value.

Proof

By Theorems 8.6 and 8.5. □

9. References

[DDS83] Dolev, D., Dwork, C., and Stockmeyer, L. "On the Minimal Synchronism Needed for Distributed Consensus", *Proc. 24th Symposium on Foundations of Computer Science*, Tuscon, Arizona, November 1983, pp. 393-402.

[DS82] Dolev, D., and Strong, H.R. "Polynomial Algorithms for Multiple Processor Agreement". *Proc. 14th ACM Symposium on Theory of Computing*, 1982, pp. 401-407.

[LSP82] Lamport, L., Shostak, R., and Pease, M. "The Byzantine Generals Problem". *ACM Trans. on Programming Languages and Systems*, vol. 4, no. 3 (July 1982), 382-401.

[LFF82] Lynch, N., Fischer, M., and Fowler, R. "A Simple and Efficient Byzantine Generals Algorithm". *Proc. Second IEEE Symposium on Reliability in Distributed Software and Database Systems*, Pittsburgh, 1982, pp. 46-52.

[PSL80] Pease, M., Shostak, R., and Lamport, L. "Reaching Agreement in the Presence of Faults". *Journal of the ACM*, vol. 27, no. 2 (1980), pp.228-234.

[Rabin83] Rabin, M. "Randomized Byzantine Generals". *Proc. 24th Symposium on Foundations of Computer Science*, Tuscon, Arizona, November 1983, pp. 403-409.

[RSA78] Rivest, R., Shamir, A., and Adleman, L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Comm. of the ACM*, vol. 21, no. 2 (Feb. 1978), pp. 120-126.

[Shamir79] Shamir, A. "How to Share a Secret". *Comm. of the ACM*, vol. 22, no. 11 (Nov. 1979), pp. 612-613.