THE BEST OF BOTH WORLDS:
GUARANTEED TERMINATION IN FAST RANDOMIZED
BYZANTINE AGREEMENT PROTOCOLS

by

O. Goldreich and E. Petrank

Technical Report #634
June 1990
The Best of Both Worlds: 
Guaranteeing Termination in Fast Randomized 
Byzantine Agreement Protocols. 

Oded Goldreich 
Erez Petrank 

Department of Computer Science 
Technion 
Haifa, Israel.

ABSTRACT 

All known fast randomized Byzantine Agreement (BA) protocols have 
(rare) infinite runs. We present a method of combining a randomized BA proto­
col of a certain class with any deterministic BA protocol to obtain a randomized 
protocol which preserves the expected average complexity of the randomized 
protocol while guaranteeing termination in all runs. In particular, we obtain a 
randomized BA protocol with constant expected time which always terminate 
within $t + O(\log t)$ rounds, where $t = O(n)$ is the number of faulty processors.

Keywords: Byzantine Agreement, Randomized Algorithms, Distributed Computing.
1. Introduction

Byzantine Agreement (BA) is a fault tolerant distributed task with the following input-output relations:

1. Agreement: For every set of inputs and any behavior of the faulty processors, all good processors should output the same value.
2. Validity: If all good processors start with the same input value, they should all output this value, no matter what the faulty processors do.

BA was introduced in the seminal paper of Pease, Shostak, and Lamport [PSL]. It has since been recognized as a key problem in the area and attracted much research. The status of BA, as well as other distributed tasks, heavily depends on the underlying communication model. In this note we assume that communication takes place in synchronous rounds and via private channels connecting each pair of processors. (Brief remarks on the asynchronous model can be found in section 6).

Let \( n \) be the number of processors and \( t \) be the number of faulty processors, [PSL] showed that no solution exists if \( t \leq 3t \). They also gave a deterministic algorithm solving the problem for the case \( n > 3t \). Their protocol consisted of \( t+1 \) rounds but required exponential amount of communication. A deterministic algorithm with polynomial message (and computation) complexity and \( t+1 \) rounds for \( t = O(n) \) was recently introduced by Moses and Waartz [MW]. Fischer and Lynch [FL] proved that \( t+1 \) rounds are optimal (i.e. every deterministic BA protocol has a run which take at least \( t+1 \) rounds). Improvement is possible through randomization.\(^1\) In particular, Feldman and Micali [FM], improving over previous ideas\(^2\) of Ben-Or [Be], Rabin [R], Chor and Coan [CC], and Bracha [Br], presented a randomized BA protocol with a constant expected time for \( t = O(n) \). However, their protocol has a shortcoming: there is no bound on the number of rounds in the worst case (i.e. for the worst sequence of coin-tosses). Namely, for every \( m \), the probability that the protocol proceeds more than \( m \) rounds is (very small, yet) greater than 0. This disadvantage is shared by all randomized BA protocols (with expected number of rounds less than \( t+1 \)) known so far.\(^3\) Is this disadvantage unavoidable? Namely, must every randomized BA protocol with expected number of rounds smaller than \( t+1 \), have unbounded runs? This question is the focus of this note.

Our main result is a randomized BA with a constant expected number of rounds in which all runs take at most \( t+O(\log t) \) rounds. In fact we show how every randomized BA protocol of certain natural class (to which almost all known randomized BA protocols belong) can be transformed into a randomized BA which always terminates. Yet has the same (up to an additive constant) expected running time as the original protocol. Hence we obtain the best of both worlds: on one hand, a randomized BA protocol with expected running time the same as the best

\(^1\) Interestingly, Randomized protocols offer no advantage in case one requires concurrent termination [CoD].
\(^2\) These works are not all in the same model. For further details see [ChD].
\(^3\) Also in the weaker model of fail-stop failures, fast randomized agreement protocols (e.g. [CMS]) have (rare) infinite runs.
possible randomized BA protocol and on the other hand, there are no infinite runs (as is the case of reasonable deterministic BA). In other words, our randomized BA protocol is "almost optimal" on the average and on the worst case simultaneously.

It should be stressed that naive attempts to solve the problem fail. In particular, one cannot just run a deterministic protocol in case the randomized protocol has not stopped within a predetermined number of rounds, since the problem of determining whether the protocol has stopped within a given number of rounds is as hard as concurrent termination (which cannot be achieved in less than \( t \) rounds \((\text{CoD})\)). The same problem occurs when one tries to run deterministic and randomized protocols in parallel, and take the output value of the protocol that finishes earlier.

2. The Model

We consider a synchronous system of \( n \) processors with unique identities, in which every pair of processors is connected via a private line. Each processor has a local input value, and has to decide upon a local output value. We distinguish between the notion of writing a local output and the local termination of a processor. When a processor gains enough information to decide upon its output, it writes the output into its output tape. In most previous works, a processor that writes its output, finishes its role in the protocol.\(^4\) We shall make each processor proceed after writing its output, for the benefit of the other processors. Therefore local termination will occur after the round in which the output is written. We stress, that a processor writes an output only once (and therefore no further writing takes place in these extra rounds). Once an output is written, the processor never tries to "change its mind".

The local running time of a processor in a given execution is the number of rounds needed until its local termination. The global running time of the protocol (in a given execution) is the maximum over the local running times of the good processors. For each possible input and adversary behavior, we can define the expected running time over the probability space of the coin-tosses. The average running time of a protocol is the maximum expected time over all possible inputs and adversary behavior. The worst running time is the running time of the protocol for the worst inputs the worst behavior of the adversaries and for the worst sequence of coin-tosses. If (good) deterministic protocols the worst (and hence average) running times are equal to \( t+1 \). In currently known randomized protocols the average time is less than \( t+1 \) (in particular [FM] has an \( O(1) \) expected running time) and the worst case running time is unbounded.

\(^4\) "Almost optimality" on the average means up to an additive constant, while in the worst case "almost optimality" means up to an additive logarithmic factor.

\(^5\) Some of the protocols ignore this question completely, thus, implicitly identifying decision with termination.
3. The Protocol:

All the known randomized protocols share a structure based on the notion of phases (consisting of a fixed number of rounds), and the notion of current value (which is the "favorite value" of the processor at the end of a phase). Most of these protocols (in particular the [FM] protocol) have the following properties:

P1. The current value of each good processor in the beginning of the first phase, is its input value.

P2. If a good processor writes output v during a phase, then at the end of that phase all good processors have v as their current value.

P3. If all good processors enter a phase with the same current value, then this value will be written on their output tape after that phase. (This value will also remain their current value).

Note that in some of the original protocols, writing the output is (implicitly or explicitly) identified with local termination. We shall explicitly mention stopping, and writing the output will not imply local termination. Our protocol can use any randomized protocol that meet the three properties, and any deterministic BA protocol.

Our Protocol (for a processor P)
The protocol will be parametrized by an integer (k ≥ 1).

\[ \text{phase} \leftarrow 0 \]

REPEAT

Run a single phase of the randomized protocol.

\[ \text{phase} \leftarrow \text{phase} + 1 \]

UNTIL reached phase k OR local writing has taken place.

CASE 1: phase < k.

Run a single phase of the randomized protocol, without any further writing.

CASE 2: phase = k AND local writing has taken place.

Participate in a deterministic protocol that starts after the kth phase. (Without any further writing!). Use your current value as an input to the deterministic protocol.

CASE 3: No local writing has occurred.

Interestingly, the randomized protocols induced by our transformation do not fall into this category.
Participate in the deterministic protocol starting after the $k$th phase and write the local output determined by this execution. The input to the deterministic protocol is the current value at the end of phase $k$.

**END**

4. Correctness

**Validity:** If all good processors start with the same value then by P1 and P3, during the first phase they will all write their input values.

**Agreement:** P2 and P3 together guarantee that, in the original randomized BA protocol, all the good processors write their output values in the same phase or during two consecutive phases. In analysing our protocol there are three possible cases to consider:

a: The first good processor to write an output, did so in phase $i < k$. In this case, all good processors wrote their values in phases $i$, $i+1$ (both smaller or equal to $k$). In these phases our protocol is still following the instructions of the randomized protocol, and therefore agreement is guaranteed by the agreement property of the randomized protocol.

b: The first good processor to write an output, did so during the $k$th phase. If all the good processors wrote their value during the $k$th phase, then we use the same argument as in case (a). Otherwise, some of the good processors wrote their values during the $4$th phase, while the others entered the deterministic protocol without having written an output. By property P2, all good processors have at this stage the same current value and those which wrote an output, wrote the same output value. Remember that though some of the good processors have written their values, they all participate in the deterministic protocol. The validity property of the deterministic protocol assures us that all the remaining good processors will write their identical current value into their output tape.

c: None of the good processors has written an output during the first $k$ phases. In this case, all the good processors write their values during the deterministic protocol, and therefore the agreement follows from the agreement property of the deterministic protocol.

We emphasize again that a processor does not write an output twice. Therefore further participation in the protocol after writing a local output, does not change the value written before.

5. Complexity Analysis

Let $r$ denote the number of rounds in a phase, $P_k$ - the probability that the randomized protocol enters the $k$th phase, (i.e., has not finished in $k-1$ phases), $T_R$ - the expected running time of the randomized protocol, and $T_D$ the running time of the deterministic protocol. (All time units

---

7 This convention is necessary since at the last phase some of the good processors might have locally terminated, leaving the other good processors in minority. These good processors have already written a value while being in majority, but now may be influenced to change their mind being in minority.
are in rounds. Clearly $T_R \geq r$.

Let $X$ be a random variable representing the running time of our protocol, and let $Y$ be a random variable describing the running time of the original randomized protocol (which may have infinite runs). Let $Z$ be a random variable that represents the additional time due to the use of the deterministic protocol. Namely, $Z = 0$ if $Y < k/r$ and $Z = T_D$ otherwise. Clearly, $X$, $Y$, and $Z$ are depending on each other. Furthermore, it is obvious that $E(X) \leq E(Y + r + Z)$ (the additive term $r$ represents the extra phase we add to the randomized protocol). Therefore:

$$E(X) \leq E(Y) + r + E(Z)$$

The worst running time of the protocol is

$$r \cdot k + T_D$$

Note that by the Markov inequality\(^8\) it is always true that

$$P_k \leq \frac{T_R}{k \cdot r}$$

Therefore, setting $k = \frac{T_D}{r}$ gives an expected running time of less than $2 \cdot T_R + r$, and worst running time of $2 \cdot T_D$. However, in many of the known randomized protocols (and in particular in the [FM] protocol) a much stronger statement holds; namely, for some constant $c < 1$,

$$P_k \leq c^{k-1}$$

Setting $k = O(\log T_D)$, we get expected running time of $T_R + r + 1$ and worst case running time of $T_D + O(\log T_D)$. In particular, let us use a standard deterministic protocol (such as in [MW]) with $T_D = r+1$, and the randomized protocol suggested in [FM] with $r=15, T_R=56$ and $P_k \leq 0.73^{k-1}$. Finally, choosing $k = O(\log t)$ gives a constant expected running time and worst case running time of $t + O(\log t)$.

The message complexity and the computational complexity of our protocol is at most the sum of the corresponding complexities of the original protocols. In fact, the average message (resp. computational) complexity of our protocol is the sum of the average complexity of the original randomized protocol and $P_k$ times the complexity of the deterministic protocol.

\(^8\) Markov inequality states that for every non-negative random variable $X$ and all $\alpha > 0$

$$\text{Prob}(X \geq \alpha) \leq \frac{E(X)}{\alpha}$$

Technion - Computer Science Department - Technical Report CS0634 - 1990
6. The Asynchronous Case

Fischer, Lynch and Peterson [FLP] proved that no deterministic solution exists for the asynchronous BA problem even with one faulty processor. It follows that there is no randomized asynchronous protocol that terminates for all coin-tosses, all inputs, and all adversary behavior. Otherwise choose in advance a sequence of coin-tosses, say - all zeros, and get a deterministic protocol that always terminates contradicting [FLP]. Therefore, in the asynchronous case it is not possible to "benefit from both worlds".

7. Conclusions and Open Problems

In light of the result presented here, even the greatest opposers of randomization must agree to use it for Byzantine Agreement in networks of substantial size. There is almost nothing to lose\textsuperscript{9}. If worse comes to worst, we end up with essentially the same complexity as in the deterministic case, whereas if we are even slightly "lucky" we gain a lot.

The choice of $k$ introduces a trade-off between worst case and average case: the worst case increases linearly with $k$, while the average case decreases exponentially with $k$. Can a better trade-off be found? In particular, can one obtain a smaller (than log $t$) addition to $T_D$ for the worst case, while keeping the expected time a constant?

8. Acknowledgements

We are grateful to Benny Chor for helpful discussions. We are also grateful to the anonymous referees for their thorough and helpful remarks.

9. References


\textsuperscript{9} As stated in [CoD], randomization cannot help when concurrent termination is necessary. We consider here only BA where there is no requirement on concurrent termination.


