Randomized Distributed Agreement Revisited

(Extended Abstract – in *Proc. FTCS '93*)

Piotr Berman * Juan A. Garay †

Abstract

Distributed Agreement (*DA*) is one of the fundamental problems in the theory and practice of fault-tolerant distributed systems. It requires correct processors (agents, players) to agree on an initial value held by one of them, despite the (even malicious) behavior of a subset of size t. A randomized solution to the problem achieves agreement with a high probability after a constant number of communication rounds.

In this paper we present a succint and efficient randomized *DA* protocol for asynchronous networks that works for $n > 5t$ processors, where n is the size of the network. The protocol has low communication complexity ($\Theta(\log n)$ message size) and does not require any cryptographic assumption. The protocol belongs to the class of protocols that require a "trusted dealer," who is in charge of a suitable network initialization, and represents an improvement in terms of number of processors to previous solutions presented in [17, 18, 20].

We contrast our approach to the class of protocols that are currently able to perform randomized agreement from scratch, an unlimited number of times (e.g., [14]), but have a communication cost that might be infeasible in many cases.

1 Introduction and Problem Statement

The *Distributed Agreement* (*DA*) problem [15] is one of the fundamental problems in the theory and practice of distributed systems. It abstracts out a wide variety of situations in which processors (agents, players), which communicate through messages, must coordinate a decision, regardless of the misbehavior of a fraction of the participants. A solution to the problem becomes more challenging and at the same time more robust when such a final consistent state can be reached while placing no restriction on the behavior of the faulty processors. The problem can be formally stated as follows.

We are given a set of processors P of which some unknown subset T, $\#T \leq t$, are *bad* or *faulty* and may exhibit arbitrary (*Byzantine*) behavior, even appearing to act in concert maliciously. Processors from $P - T$ are called *good* or *correct*. Every processor is given an initial value, 0 or 1.[1] The goal is to design a protocol (i.e., an algorithm for each of the participants) such that upon termination, the final values of the correct processors satisfy the following two conditions:

*Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. berman@cse.psu.edu.

†Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974. garay@research.bell-labs.com.

[1] To concentrate on binary agreement is no restriction. Given a solution to this version of the problem, the design of multi-value solution is straightforward (see, e.g., [17]).

- **Agreement:** They are all equal.
- **Validity:** They are all equal to the initial value, if the latter is unique.

We use the standard model of a completely connected network of processors numbered from 1 to n. The computation performed by the network evolves as a series of *rounds*, during which the processors send messages, receive messages, and perform local computations according to their algorithm. There already exist many protocols for this problem because there are several quality parameters that can be optimized: the total number of processors n, as a function of t (so-called *resiliency*); the number of rounds of communication r; and the computation and communication complexity, the latter given by, e.g., the maximal message size m.

An obvious lower bound for the message size is $m = 1$. Under the *synchronicity* assumption (i.e., the existence of a global clock), it is known that the minimum number of processors for the problem to be solvable is $n > 3t$ [15]. Also under this assumption, a linear lower bound in the number of communication rounds ($r \geq t + 1$) has been established for this problem even if no failures actually occur [9]. Thus, overcoming potential faults is expensive, even though, one might argue, they are rarely observed in practice. The situation gets worse if the synchronicity assumption is dropped. In particular, it has been shown by Fischer *et al.* [12] that in a completely asynchronous environment, no *deterministic* solution to the problem exists even in the presence of only one fault.

This has led researchers to explore ways in which the price for fault tolerance may be reduced. *Randomization* has proven fruitful in circumventing the $t + 1$ lower bound on round complexity, as well as the [12] impossibility result. In a randomized solution to the agreement problem, each processor can use the results of random coin tosses in the course of reaching agreement. In this setting, protocols that run for an *expected constant* number of rounds are possible, and an $n > 3t$ bound has been derived for the randomized environment as well [16].

In this paper, we are interested in robust and efficient solutions to the agreement problem. Our first requirement forces us to give up the synchronicity assumption—although many practical computer systems use synchronous coordination protocols, we would not like our predisposition to tolerate arbitrary faults be weakened by alternate assumptions. The second, to look for solutions with low computation and communication complexity costs, while requiring (expected) constant time to termination. Otherwise, the performance of any protocol with even low polynomial message size (e.g., $m = O(n^3)$) will be seriously retarded as the size of the system grows moderately. In particular, in this paper we prove the following result:

Theorem 1 *There exists a randomized agreement protocol for an asynchronous network that terminates in an expected constant number of rounds and uses messages of size* $\Theta(\log n)$, *provided* $n > 5t$.

Our protocol represents an improvement in terms of resiliency to the one presented in [17]. As in [18, 17, 20], the protocol uses the notion of a "trusted dealer" and predistributed pieces of a coin toss. The improvement is achieved by a minor modification to the protocol and improved analysis. The main virtues of our result are its simplicity and low communication volume and message size.

The balance of the paper is organized as follows. In Section 2 we briefly review the use of randomization in *DA*, and contrast our result with other work. In section 3, we review the so-called *secret sharing* techniques, used by our solution, while the randomized agreement protocol is presented in Section 4. We conclude with some final remarks.

2 Randomization in Distributed Agreement

Randomization for reaching *DA* was first used by Ben-Or [4], and then perfected by Rabin [18], using the idea of a *common coin*, a tool that has been essential to all subsequent solutions. The main idea is that in the course of reaching agreement, each processor can use the outcome of random coin tosses, in case they are not "overwhelmingly convinced" of a unanimous state of things.

The randomized variant of the agreement problem has received a lot of attention in the last few years; in particular, much progress has been achieved in the synchronous setting [5, 6, 13, 10], where the synchronicity of the system allows for the construction of an "internal" coin-tossing mechanism, which requires a given number of preprocessing rounds. After these rounds, agreement can be reached in *constant* expected time, an unlimited number of times.

More recently, a break-through technique proposed by Feldman and Micali [14] allows agreement to be reached in constant expected time from scratch, i.e., without any preprocessing. This technique can also be applied to the asynchronous situation; their protocol, however, performs simultaneously many broadcast operations, thus requiring a high volume of communication—$\Omega(n^3)$ messages per communication link—and messages of very large size ($\Omega(n^3)$). Under these circumstances the performance of any protocol is seriously retarded with the increase of the number of processors.

In contrast, Rabin's approach [18], and subsequent protocols of Toueg [20], Perry [17] and our protocol of Section 4, give fast agreements once the network has been suitably initialized. The approach uses the notion of a "trusted dealer," who is in charge of predistributing pieces of a common coin toss. These pieces are then put together using the concept of *reconstructible secret sharing* [1]. Admittedly, this approach yields solutions that are "expendable" in the sense that intervention will be required from the trusted dealer once the coin tosses have been exhausted. We view this tradeoff of communication costs vs. number of agreement instances as a challenging open problem.

Regarding our solution, the protocol represents an improvement in terms of resiliency to the one presented in [17] (from $n > 6t$ to $n > 5t$), and it doesn't use cryptography (more than in what it is required to achieve authenticated links), nor many broadcast operations. Toueg's [20] algorithms are $n/3$-resilient, but at the cost of using both. If cryptography may be used, our protocol also tolerates $< n/3$ faults. More specifically, we show in Section 4 that there exists a probabilistic protocol that, in an asynchronous setting with $n > 5t$, terminates after R rounds with a probability of reaching *DA* of at least $1 - 2^{-\frac{R-1}{2}}$, using messages of size $\Theta(\log n)$.

3 Reconstructible Secret Sharing

The idea of *secret sharing* was introduced by Shamir [19], and independently by Blakely [3].[2] A dealer privately sends a *piece* of a secret message to each processor in a network of n processors, at most t of which are bad. Assuming $n \geq 2t + 1$, any piece by itself is useless; however, any collection of $n - t = t + 1$ pieces may be used to recover the secret. Therefore, the secret may not be recovered without the cooperation of at least one correct processor. On the other hand, the correct processors can recover the secret without any help from the bad processors.

Shamir's proposal for sharing a secret s goes as follows. Let $p > n$ be a prime; all arithmetic takes place in Z_p, a finite field of p

[2] The reader is referred to [8] for a more up-to-date insight on secret sharing.

elements. The dealer chooses a random polynomial S of degree t, whose constant term $S(0) = s$, the secret to be shared.[3] Every processor i receives $S(i)$, the value of S evaluated at its identity. To recover s, every processor sends its value to all the other processors, $S(0)$ can then be obtained using polynomial interpolation, for which $t+1$ points are needed.

The dealer chooses S randomly, so the pieces of the bad processors are uniformly distributed, and since any set of $t+1$ values is consistent with a polynomial of degree t, the value of the secret remains uniformly random from the point of view of the bad processors until a piece is revealed by a good processor. (Thus this scheme is so-called t-*private* [1].)

The main problem with the scheme presented so far is the so-called *dirty pieces* problem: The interpolation used to recover s requires that all the points of the polynomial be correct; if a single value is incorrect, then the interpolation will fail to produce the right result. The bad processors can then interfere by sending wrong values. When the number t of bad processors is a constant fraction of the n, then a random subset of size $t+1$ contains a bad processor with overwhelming probability. Thus, the reconstruction of the correct polynomial (and secret) poses a non-trivial problem for the good processors.

If cryptography may be used [18, 20], then *signatures* could overcome the problem of dirty pieces. This solution, however, increases the volume of the secret to be distributed by a factor larger than n, and it also leads to involved computation. We therefore prefer an approach suggested by Ben-Or *et al.* [1] based on standard error-correcting code (ECC) techniques [2]. Due to the asynchronicity of the model, a good processor can only wait to receive $n-t$ pieces of the secret, since otherwise the bad processors could delay the execution of the protocol forever. The correct processor must then be able to detect and correct up to $2t$ errors/wrong values: At most t values from the bad processors plus t values it can't wait for (the so-called *erasures* in the ECC jargon).

Reed-Solomon ECC's [2] provide a way to recover efficiently the correct polynomial in the presence of t errors and t erasures, provided that $n-t$ values of the t-th degree (*information*) polynomial are considered. This leads to a *Reconstructible Secret Sharing* (RSS) protocol which does not need signatures, and which works for $n > 5t$.

Therefore the choice of S with degree t allows a correct processor to reconstruct the secret $S(0)$ with t missing values (erasures) and with at most t wrong values in the remaining $n-t$ pieces of the secret. In the next section RSS is employed in the construction of the *DA* protocol.

4 The Randomized Distributed Agreement Protocol

In this section we describe TRTL (for *Two-Rounds-is-Too-Late*), a non-cryptographic, low-communication complexity agreement protocol that works in asynchronous networks, provided $n > 5t$. TRTL proceeds as a repetition of *phases*, a phase in turn consists of three rounds of message exchange. The *configuration* of a phase is the initial multiset of values of the correct processors. The first configuration consists of the initial values; subsequent configurations consist of the values adopted in the previous phases. We now introduce the concept of the *bias* of the configuration at a given phase l, b_l, which, intuitively, represents the popularity of a given value among the correct processors. More specifically, we say that the bias is *well defined* if there exists a value v such that v occurs at least $n-3t$ times among the $n-t$ correct processors. If $b_k \neq 0$ and $b_k \neq 1$, then $b_k = \perp$. If a value

[3] The probability that the t-th coefficient of S is 0 is $1/p$; therefore "degree t" means "at most t."

occurs $n - t$ times, then the configuration is said to be *unanimous*. Note that it is not possible for $b_l = 0$ and $b_l = 1$ to hold simultaneously, otherwise $2 \cdot (n - 3t) \leq n - t$, and consequently, $n \leq 5t$. Protocol TRTL is presented in Figure 1.

In Exchange 1, all processors broadcast their current value of V, and then wait to receive similar messages from $n - t$ processors (in all three exchanges, we include in the count of messages the message that the processor sends to itself). At this point, correct processors make a preliminary decision for V's new value, depending on the multiplicity of the values received. Exchange 2 is crucial to avoid that correct pieces of the secret be revealed to early. If this were the case, then under certain circumstances the bad processors could manipulate the configuration and change its bias in such a way that disagreement could persist forever.

This is what can happen if (a piece of) the secret is revealed too early: suppose a subset—call it *DE* for deciding early—of $n - 2t$ processors always finish sending their values before the other t correct processors (group these late-comers into a set called *DL*—deciding late). Only *after* the t bad processors see the values supplied by *DE* and *DL*, do they broadcast their values. If the values provided by the processors in *DE* and *DL* are divided in a certain way, then by sending different messages to different processors and by choosing which proper messages to deliver to the processors in *DL* (i.e., they can choose to deliver some messages from *DE* and some from *DL*), the bad processors have the following power: They can choose which processors are forced to choose the bias and which must choose the secret.

I.e., the bad processors have the ability, given some initial scenarios, to decide what a good processor does. By knowing the secret *before* deciding which messages to send, and if the secret is different from the initial bias, then the bad processors can effectively choose the bias of the configuration for the next phase by causing some correct processors to be forced to the initial bias and causing the others to choose the secret. Thus, the bad processors can prevent agreement **forever**. (What the proof of part 2 of Theorem 1 says is that the bad processors can't discover the secret until it is too late for them to affect the outcome.)

```
V := v_i; (* i's initial value *)
for k := 1 to R do begin

   (* Exchange 1: polling and early dec. *)
   send(V) to every processor;
   collect n − t values;
   C := the multiplicity of most popular
   value (say v);
   if C ≥ n − 2t then
      V := v
   else
      V := ⊥;

   (* Exchange 2: readiness *)
   send('ready') to every processor;
   collect n − t 'ready' messages;

   (* Exchange 3: lottery and late dec. *)
   send(s_{k,i}) to every processor;
   collect n − t pieces;
   RSS; (* s_k is the current secret's value *)
   if V = ⊥ then
      V := s_k;

end;
```

Figure 1: TRTL, the randomized agreement protocol; code for processor i.

Finally, in Exchange 3 correct processors reveal their piece of the secret to everybody, and collect $n - t$ pieces from other processors. We let $s_k \in \{0, 1\}$ denote the value of the secret for phase k, and $s_{k,i}$ denote processor i's piece of the secret. Using the Reconstructible Secret Sharing technique from the previous section the reconstruction of the proper secret is possible given that $n > 5t$, and that there are t missing pieces and at most t wrong pieces. We also note that ev-

ery message must include the phase and exchange step number in which it is sent. For clarity, these are omitted from the description of the protocol in Figure 1.

Before proving the correctness of TRTL, we prove a series of technical lemmas. From now on we assume $n > 5t$.

Lemma 1 *Let b_k be a well-defined bias. Then a correct processor takes a value v during Exchange 1 of phase k only if $v = b_k$.*

Proof Suppose a processor i sets its value to v during Exchange 1 of phase k. Then it received value v from at least $n - 2t$ processors during that exchange. Since at most t of them are bad, the configuration contains v at least $n - 2t - t$ many times. Hence, $b_k = v$. □

Lemma 2 *Assume that in phase k, either $b_k = \perp$, or b_k is well defined and $s_k = b_k$. Then at the end of phase k the configuration is unanimous.*

Proof According to Lemma 1, a correct processor either decides on b_k early (in Exchange 1) because it collected b_k $(= s_k)$ $n-2t$ times, or late (in Exchange 3). I.e., no correct processor can take $(1 - s_k)$ as its next value. □

Definition 1 *A phase k is* bad *if the following two conditions are simultaneously satisfied:*

- *At the beginning of the phase, the configuration is not unanimous.*
- $b_k = 1 - s_k$.

A phase is *good* otherwise.

We note that when phase k is good, then by the previous lemmas the probability of unanimity (Agreement) at the end of the phase is 1. Let p_k denote the probability that phase k is bad. The following lemma establishes the probability that these least favorable conditions for agreement carry on to subsequent phases.

Lemma 3 *Assume that phase k is bad, then the probability that phase $k+2$ is bad is at most* 0.5. *I.e., $p_{k+2} \leq 1/2 \cdot p_k$.*

Proof [Sketch] If phase k is bad, two cases are possible:

1. The bias b_{k+1} is set before the bad processors learn s_{k+1}. Then either $b_{k+1} \neq \perp$ and there is 1/2 chance that $s_{k+1} = b_{k+1}$, or $b_{k+1} = \perp$. Therefore the probability that phase $k+1$ is bad is $p_{k+1} \leq 1/2 \cdot p_k$.
2. The bad processors learn s_{k+1} *before* setting bias b_{k+1}. Then no value could be taken by more than $n - 3t - 1$ correct processors before they learn s_{k+1}. All the bad processors need to reconstruct the secret is one extra piece from a good processor. Let f be the first correct processor that reveals its piece of the secret (first action of Exchange 3). f does so after receiving $n - t$ 'ready' messages in Exchange 2, therefore at least $n - 2t$ correct processors already collected $n - t$ values at Exchange 1 of phase $k + 1$. Because the bias of phase $k + 1$ is not yet set, all those processors will take s_{k+1} as their value. Since there is a 1/2 chance that $s_{k+1} = s_{k+2}$, the probability that phase $k + 2$ is bad is then $p_{k+2} \leq 1/2 \cdot p_k$.

We finally note that in either case it also holds that $p_k \geq p_{k+1}$, for any k. This completes the sketch of the proof. □

We are now ready to prove our main result, which we re-state in an equivalent form, in light of this section's definitions. The communication complexity of the protocol is immediate.

Theorem 1 *In an asynchronous network with $n > 5t$ and initial values $v_i \in \{0,1\}$, $i = 1, ..., n$, protocol* TRTL *terminates (in R phases) satisfying:*

1. *If the configuration is unanimous with initial value v, then for every correct processor $V = v$.*
2. *If the configuration contains different initial values, then with probability at least $1 - 2^{-\frac{R-1}{2}}$ all correct processors have the same final value. In particular, we prove that at each phase k, there is a 0.5 probability of agreement after phase $k+2$.*

Proof [Sketch]

1. If the configuration is unanimous (i.e., $v_i = v_j = v$ for i, j correct processors), then every correct processor collects v at least $n - 2t$ times and decides on v at the end of Exchange 1. Consequently, once agreement on some value v is reached by the correct processors, it will hold at each subsequent iteration (*Persistence of Agreement*).
2. The configuration contains different values, in which case we can distinguish the following sub-cases:
   (a) b_k is well defined and $s_k = b_k$, or $b_k = \perp$. Then the claim follows from Lemma 2.
   (b) b_k is well defined and $s_k \neq b_k$. Phase k is bad. By Lemma 3, $p_R \leq (1/2)^{\frac{R-1}{2}}$; thus, the probability of unanimity after phase R is at least $1 - p_R$.

□

The resiliency improvement of our protocol to the one presented in [17] is achieved by introducing a "readiness" message exchange, tantamount to requiring that the secret be shared by $n - t$ processors, instead of $t + 1$. With $n > 5t$, the bad processors have now the power to learn the next phase's secret and manipulate the bias of the configuration accordingly, but they are not able to do so with the secret (and bias) of two phases ahead. Here is where the power of randomization wins again. We note that the correctness of TRTL still holds if the requirement of $n - t$ pieces of the secret is relaxed to $n - 2t$.

5 Final Remarks

In this paper we have presented a succint and efficient randomized Distributed Agreement protocol for asynchronous networks that works for $n > 5t$ processors. The protocol has low communication complexity ($m = \Theta(\log n)$) and does not require any cryptographic assumption. The protocol belongs to the class of protocols that require a "trusted dealer," who is in charge of a suitable network initialization. On the other hand, the class of protocols that are currently able to perform randomized agreement from scratch, an unlimited number of times (e.g., [14]), have a communication cost that might be infeasible in many cases. In a recent development, Canetti and Rabin [7] present a protocol that improves on the resiliency of [14] to the optimal (i.e., $n > 3t$) while maintaining similar communication overhead, at the cost of being non-error-free. We are thus left with the intriguing dichotomy of "expendable" vs. high-communication cost solutions to the randomized agreement problems.

References

[1] M. Ben-Or, S. Goldwasser and A. Wigderson, "A non-cryptographic completeness theorem for multi-party protocols," *Proc. 20th ACM Symp. on Theory of Computing,* pp. 1-10, May 1988.

[2] R. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Pub. Co., 1983.

[3] G. Blakely, "Safeguarding Cryptographic Keys," *Proc. AFIPS,* Vol. 48, pp. 313-317, NCC, June 1979.

[4] M. Ben-Or, "Advantage of Free Choice: Completely Asynchronous Agreement Protocols," *Proc. 2nd Annual ACM Symp. on Pples. of Distributed Computing,* pp. 27-30, August 1983.

[5] G. Bracha, "An $O(\log n)$ Expected Rounds Randomized Byzantine Generals Algorithm," *Proc. 17th ACM Symp. on Theory of Computing,* pp. 316-326, May 1985.

[6] B. Chor and B. Coan, "A Simple and Efficient Randomized Byzantine Agreement Algorithm," *IEEE Transactions on Software Engineering,* Vol. SE-11, No. 6, pp. 531-539, June 1985.

[7] R. Canetti and T. Rabin, "Fast Asynchronous Byzantine Agreement with Optimal Resilience," to appear in *Proc. 25th Annual ACM Symp. on Theory of Computing*, San Diego, CA, May 1993.

[8] C. Dwork, "On Verification in Secret Sharing," IBM Research Report RJ 8406, October 1991.

[9] D. Dolev and H.R. Strong, "Authenticated Algorithms for Byzantine Agreement," *SIAM Journal of Computing*, Vol. 12, pp. 656-666, 1983.

[10] C. Dwork, D. Shmoys and L. Stockmeyer, "Flipping Persuasively in Constant Expected Time," *Proc. 27th Annual Symp. on Foundations of Computer Science,* pp. 222-232, October 1986.

[11] M.J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," Yale University Tech. Report *YALEU/DCS/RR-273*, 1983.

[12] M.J. Fischer, N. Lynch and M. Paterson, "Impossibility of distributed consensus with one faulty process," *JACM,* 32(2), pp. 374-382, 1985.

[13] P. Feldman and S. Micali, "Byzantine Agreement in Constant Expected Time (and Trusting No One)," *Proc. 26th Symp. on Foundations of Computer Science,* pp. 267-276, October 1985.

[14] P. Feldman and S. Micali, "Optimal Algorithms for Byzantine Agreement," *Proc. 20th ACM Symp. on Theory of Computing,* pp. 148-161, May 1988.

[15] L. Lamport, R.E. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems,* Vol. 4, No. 3, pp. 382-401, July 1982.

[16] A. Karlin and A. Yao, "Probabilistic Lower Bounds for Byzantine Agreement," unpublished manuscript, 1986.

[17] K.J. Perry, "Randomized Byzantine Agreement," *IEEE Transactions on Software Engineering,* Vol. SE-11, No. 6, pp. 539-546 June 1985.

[18] M. Rabin, "Randomized Byzantine Generals," *Proc. 24th Symp. on Foundations of Computer Science,* pp. 403-409, Nov. 1983.

[19] A. Shamir, "How to share a secret," *Communications of the Association for Computing Machinery,* Vol. 22, pp. 612-613, Nov. 1979.

[20] S. Toueg, "Randomized Byzantine Agreement," *Proc. 3rd ACM Symp. on Principles of Distributed Computing,* pp. 163-178, Canada, August 1984.