Simple Constant-Time Consensus Protocols in Realistic Failure Models

(Extended Abstract)

Benny Chor[1] Michael Merritt[2] David B. Shmoys[3]

MIT, Cambridge, MA
AT& T Bell Labs, Murray Hill, NJ and MIT, Cambridge, MA
Harvard University, Cambridge, MA

Abstract: Using simple and elegant protocols, we show how to achieve consensus in constant expected time, within realistic failure models. Significantly, the strongest models considered are completely asynchronous. A nearly matching lower bound is also given.

1. Introduction

Randomization has proved to be an extremely useful tool in the design of protocols for distributed agreement. In this paper we present new randomized protocols for the consensus problem in synchronous and asynchronous fail-stop and failure-by-omission models. These protocols all terminate within constant expected time, and unlike previous efficient protocols, are very simple and need not rely on any preprocessing. The major novelty of our algorithms is the notion of a weak form of a global coin, and a method for generating it.

We define the *consensus problem* as follows: processor i has a private binary value v_i; at the termination of the protocol all processors have agreed on a common value v; if all v_i were equal initially, the final value agreed upon is this common value.

We shall initially consider the following synchronous model. We are given a system of n processors that can communicate through a completely connected network. The processors act synchronously, where at each time step each processor can broadcast a message, receive all incoming messages, and perform some private computation (possibly involving coin tossing). In the absence of failure, any message sent at time i will be received at time $i+1$. As a result, we will view the computation as occurring in rounds.

The situation for deterministic algorithms for consensus is well understood. A result of Dolev and Strong implies that in a synchronous fail-stop model, at least $t+1$ rounds are needed, in the worst case, to achieve consensus; they also provided an algorithm that achieved this

---

[1] Research supported in part by an IBM graduate fellowship.

[2]Research supported in part by ONR under N00014-85-K-0168, by OAR under DAAG29-84-K-0058, by NSF under DCR-8302391, and by DARPA under N00014-83-K-0125.

[3]Research supported in part by the NSF under DCR-8302385.

bound and transmits only a polynomial number of messages [DS]. In the asynchronous case, Fischer, Lynch and Paterson showed that no protocol exists for consensus in the fail-stop model which tolerates even a single fault [FLP].

Fortunately, randomization can overcome this inherent intractability. Ben-Or gave a protocol for asynchronous consensus that tolerates up to $n/2$ faults in the fail-stop model, and terminates with probability 1 [Be]. (Results of a similar nature were given by Bracha and Toueg [BT].) Unfortunately, the expected number of rounds needed to reach agreement is exponential in the asynchronous case (and can be shown to be $O(\frac{t}{\sqrt{n}})$ in the synchronous one). Rabin gave a different protocol that uses a *global* coin flip, so that each processor can use the outcome of a common coin, and the expected number of rounds is $O(T(n))$, where $T(n)$ is the time required to flip the coin. In order to implement his global coin, Rabin required some predealt information to be distributed by a trusted third party. Bracha, using a beautiful "bootstrapping" construction, showed that Rabin's result could be improved to $O(T(\log n))$ rounds. Recently it has been shown how to use cryptographic techniques to implement such an unbiased, provably-secure coin in $T(n) = O(n)$ rounds, so that overall, Bracha's procedure can be run in $O(\log n)$ expected time [Y2,ABCGM]. This $O(\log n)$ bound is the best known for Byzantine fault model without predealt information. Since our algorithms for omission faults run in constant expected time, current results leave a $\log n$ separation between the Byzantine and omission fault models.

In this paper we present protocols for achieving consensus in completely connected networks that can tolerate as many as $t = \theta(n)$ omission faults of various types. The algorithms that we present are based on producing a coin that is essentially global. We can relax the condition that each processor's view of the coin must always be identical, and in fact, the coin may even be somewhat biased. More precisely, we define a *weakly global coin* as a coin where, for both possible outcomes, at least $\lfloor n/2 \rfloor + t + 1$ processors have a common view of the outcome with constant probability. If this many processors see the same outcome, then a majority of the processors ($\lfloor n/2 \rfloor + 1$) will use this value in the consensus protocol, and reach consensus in a few more rounds. The essence of our procedure is to randomly select a temporary leader, and then to use the leader's local coin flip for the given round. After showing how such a coin can be produced in a variety of omission fault models, we then indicate how to use it to acheive consensus.

The design strategy of our protocols reflects a heuristic rule prevalent in distributed protocol design: it should be possible for simpler algorithms to defeat weaker adversaries. In the search for provably good algorithms that are also useful in practise, this rule suggests that some complex protocols have simple counterparts in more realistic fault models. In the case studied here, the algorithm against the adaptive adversary is transparent in comparison to the protocol for the Byzantine case that results from the combined work in [Br] and [ABCGM].

Finally, we show that these results are nearly tight, by showing that for any protocol for the fail-stop model, if t processor faults are tolerated, then the probability that all correct processors have decided after k rounds ($k \leq t$) is at most $1 - \frac{1}{2} \cdot \left(\frac{t}{2nk}\right)^k$. (The same result was obtained independently by Karlin and Yao [KY].) By comparison, our protocol achieves $1 - \left(\frac{2e-1}{2e} + \frac{t}{2en}\right)^{k/2}$.

2. Failure Models

Correctness proofs for fault-tolerant algorithms have a game theoretic character. They argue that the algorithms behave appropriately, even when the faults are being caused by an intelligent adversary. The capabilities attributed to this adversary have a profound effect on the design of algorithms meant to defeat it. Indeed, there are cases in which no algorithm is capable of defeating sufficiently powerful adversaries [PSL, FLP].

In Byzantine fault models, the adversary can control the behavior of some processors, causing them to send arbitrary messages whenever it likes. Such an adversary is extremely powerful, and defeating it seems to require complex and expensive algorithms. If one is modeling physical failures (as opposed to intentional

attacks), such an adversary may be unrealistically powerful.

Consider the following example. On October 27, 1980, the ARPANET suffered a catastrophic failure as the result of hardware failures in two processors. Two spurious messages were generated that brought down the whole network for a period of several hours. Clearly, the network protocols were not capable of surviving even a small number of Byzantine faults. Instead of changing the protocols, hardware error-detection was added in the next generation processors, reducing the likelihood of repetition of this Byzantine failure to an extremely small probability [Ro]. Rather than implementing protocols to defeat a Byzantine adversary, the network designers effectively chose to weaken the adversary.

The new ARPANET implementation might be best described by an omission fault model, in which processors never send spurious messages, but some messages may fail to arrive at their destination. The adversary is thus limited to specifying which messages will be delivered to their destination, and which will not. The failure models we consider here are variants of failure by omission.

For deterministic protocols, an adversary, causing failures to produce the worst possible performance, can determine the outcome of a strategy in advance. With randomization, this is no longer possible, so that it may be advantageous for the adversary to decide its strategy adaptively, as random bits are generated and used. Therefore, in modeling the power of the adversary it is crucial to specify the extent to which the adversary is adaptive, and the information it has available to determine its strategy. We will consider three limitations on the adaptiveness of the adversary.

Static faults: Throughout the life of the system, messages sent by at most t processors fail to reach their destination on time (within the round they are sent). Static faults include the fail-stop model as a special case. Most previous work on omission fault models has focused on this type of fault.

Dynamic-broadcast: During each round, messages sent by at most t processors fail to reach their destination (but this may happen to a different t processors each round). These models are more general than static fault models. They are similar to models studied in [Pi].

Dynamic-reception: Each processor receives all but at most t messages sent to it during every round (so that, if all processors broadcast every round, each processor receives at least $n-t$ messages). However, any two processors may fail to hear from a different set of t others. These models are more general than dynamic-broadcast models, and are similar to the models we will use for the asynchronous case.

We present algorithms for dynamic-broadcast and dynamic-reception models. Because these models are more general than the fail-stop or static models, our algorithms will work in these cases as well.

In addition to the limitations on the adaptiveness of the adversary mentioned above, we consider two different limitations on the knowledge available to the adversary in determining its strategy.

Message-oblivious: The adversary's choice of failure (which messages will not be delivered) is independent of the contents of the messages.

Message-dependent: The adversary is limited to polynomial resources (time and space), but its choice of failures may depend on the contents of the messages.

Finally, we will assume that the adversary has full knowledge of the hardware and software running at each processor and of the communication over the network (subject to the limitations above), but does not know the local state of the individual processors during execution (which may depend on the outcome of local coin tosses not observed by the adversary). For each combination of adaptiveness and knowledge constraints, we present an algorithm to achieve consensus in constant expected time.

3. The Message-Oblivious Case

In this section we show how to toss a weakly global coin in message-oblivious models. For the dynamic-broadcast failure model, the coin will have the property that for each outcome (heads or tails), there is some constant probability of that outcome being received by *every* processor. For the dynamic-reception failure model, there is some constant probability that for each

outcome, at least $\lfloor n/2 \rfloor + t + 1$ processor will recieve that outcome (provided t is slightly less than $n/4$).

The algorithm is perhaps the most natural one. A leader randomly volunteers, and this leader tosses a coin. More precisely, consider the following algorithm: the procedure LEADER produces a local biased bit where the probability of a 1 ("I volunteer") is equal to $\frac{1}{n}$; the procedure RANDOM BIT produces a local unbiased bit.

Code for processor P:

1. **function** COIN TOSS:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast (c_p, l_p)
5. receive all (c, l) messages
6. **if** a unique message with $l = 1$ was received
7. **then** COIN TOSS $\leftarrow c$ of that message
8. **else** COIN TOSS $\leftarrow 0$

Theorem 1: *The function* COIN TOSS *produces a weakly global coin in the dynamic-broadcast message-oblivious fault model, where the constant probability for either common outcome is at least* $\frac{1-\beta}{2e}$, *provided* $n > \beta t$ *(where* $\beta > 1$ *is a constant).*

Proof: In a single execution of COIN TOSS, the probability that exactly one processor volunteers is

$$\binom{n}{1} \cdot \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{e} .$$

In the analysis, we will restrict attention to executions of the procedure when this event happens. How can the adversary thwart a good coin toss? Only by preventing the leader's message from getting to all other processors. However, he must select the set of at most t faulty processors with no information about the random bits, so that if the leader is not among those picked by the adversary, its messages will reach all processors. Hence all the messages of the leader reach their destination with probability at least $\frac{n-t}{n} \geq 1 - \beta$. The second coin toss of the leader was made independently of the above conditions, and the probability of each outcome is the same. Putting the pieces together, we get the claimed bounds. □

Remark: While it suffices to require $\beta > 1$ in order to achieve a weakly global coin, we actually will require $\beta \geq 2$. This requirement is needed in the consensus protocol (see section 6).

Theorem 2: *The function* COIN TOSS *produces a weakly global coin in the dynamic-reception message-oblivious fault model, when t is slightly less than* $n/4$. *If* $t = n(1/4 - \varepsilon)$, *the constant probability for either common outcome is at least* $\alpha/2e$, *where* α *is approximately* $\frac{8\varepsilon}{4\varepsilon+1}$.

Proof: Once again, we focus on the case that exactly one processor volunteers, which happens with probability at least $\frac{1}{e}$. Recall that in the dynamic-reception fault model, every processor receives at least $n - t$ different messages each round, but different processors may receive messages from different sets of senders. Recall also, that the conditions for a weakly global coin only require that at least $\lfloor n/2 \rfloor + t + 1$ processors agree on the outcome of the global coin. This happens if the leader succeeds in reaching this many processors. Accordingly, call processors whose messages are received by $\lfloor n/2 \rfloor + t + 1$ processors *persuasive*. Since the failures are chosen independently of the identity of the leader, it is enough to show that a constant fraction, α, of the processors are persuasive during each round. Since everyone receives at least $n - t$ messages, the number of messages received is at least $n(n - t)$. Assume exactly αn processors are persuasive–then the number of messages received is at most $\alpha n^2 + (n - \alpha n)(\lfloor n/2 \rfloor + t)$. This number is achieved if each persuasive processor has n messages received, and the rest lack only 1 message received to be persuasive themselves. From $n(n - t) \leq \alpha n^2 + (n - \alpha n)(\lfloor n/2 \rfloor + t)$, we derive $\frac{\lceil n/2 \rceil - 2t}{\lceil n/2 \rceil - t} \leq \alpha$. The number of faults, t, must be such that α is forced by this relation to be a positive fraction. This occurs when $t < n/4$. In particular, when $t = n(1/4 - \varepsilon)$, α is about $\frac{1/2-(1/2-2\varepsilon)}{1/2-(1/4-\varepsilon)} = \frac{8\varepsilon}{4\varepsilon+1}$.

Thus there is at least probability $1/e$ that there is a single leader, for each value (heads or tails), there is probability $1/2$ that the leader's other coin has that value, and there is at least α probability that the leader is persuasive. By the message oblivious assumption, all these events are independent so that overall the probability of each outcome is at least $\alpha/2e$. □

It is critical to the correctness of this protocol that the adversary's choice of messages delivered each round be independent of the contents of the messages. A stronger adaptive adversary might simply check each message as it is sent; if the processor is a potential leader (its message is $(b, 1)$) then the adversary blocks the message. This stronger adversary can also be defeated, as long as the contents of the messages are unintelligible to him. In this case, any attempt at blocking the leader's message is still an essentially random act, because the adversary cannot understand the messages. This suggests that encryption could be a useful tool in designing a protocol that can defeat a more powerful adversary.

4. The Message-Dependent Case

In this section we show how cryptographic techniques can be used to toss a weakly global coin in the presence of an adaptive adversary using a message-dependent strategy. We prove that if the adversary can block the weakly global coin, then it can break the cryptosystem. Therefore, if we assume that the cryptosystem is secure, and that the adversary is limited to polynomial computing resources, then it cannot prevent consensus within constant expected time.

Let E be a probabilistic encryption scheme that hides one bit [GM]. The scheme E can be based on any trapdoor function [Y1]. As an example, the familiar RSA cryptosystem can be used, with 0 encrypted by $E(x)$, where x is chosen at random among all numbers in Z_N with least significant bit 0, and 1 encrypted by $E(x)$, where x is chosen at random among all numbers in Z_N with least significant bit 1 [ACGS]. (For this example, we assume that RSA is hard to crack.) The main theorem of this section is based on the following assumption:
(*) The encryption function E cannot be inverted in random polynomial time.

We first make the assumption that all processors use the same public key E whose decryption key they all hold (but the adversary has no access to). At the end of this section we indicate how this assumption can be removed, at some expense in the number of faults tolerated.

The only modification to the algorithm of the previous section is to replace the broadcasting of (c, l) (line 4 of the COIN TOSS function) by the broadcasting of $(E(c), E(l))$. The modified code is now:

Code for processor P:
1. **function** COIN TOSS:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast $(E(c_p), E(l_p))$
5. receive and decrypt all (c, l) messages
6. **if** recieved a unique message with $l = 1$
7. **then** COIN TOSS $\leftarrow c$ of that message
8. **else** COIN TOSS $\leftarrow 0$

We now prove that the new protocol is as hard to break as the cryptosystem it uses.

Theorem 3: *Under the assumption* (*), *the modified function* COIN TOSS *produces a weakly global coin in the message-dependent fault models, provided* $n > \beta t$ *for the dynamic-broadcast case, and* $n > 4t$ *for the dynamic-reception case. The probabilities of each outcome are as in Theorems 1 and 2, respectively.*

Proof Sketch: We will prove the result for the static and dynamic-broadcast model; the proof for the dynamic-reception model is very similar. We will again restrict attention to the event that exactly one processor volunteers, and that its random bit is 1 (the case of 0 is handled identically). This event occurs with probability at least $1/2e$. Suppose the adversary can block some of the messages of the leader with probability $\geq \frac{1}{2} + \varepsilon$ (where $\varepsilon > 0$). We show that in fact the adversary has the power to *distinguish* between the encryption of (RANDOM BIT, 0) and the encryption of $(1, 1)$. (An adversary can distinguish between two outcomes if there exists an $\varepsilon > 0$ such that the difference between the probabilities of the outcomes is at least ε.) Using a theorem of Goldwasser and Micali [GM], this leads to a polynomial time algorithm for the adversary to invert E.

The proof consists of two parts. The first part is that the modified COIN TOSS function is a zero knowledge protocol [GMR]. This basically means that the adversary can simulate polynomially many executions of this function by itself, without having any secret information,

with the same probability distribution. Therefore previous executions give the adversary no information that it cannot get by itself.

The second part of the proof is done by a protocol simulation. Suppose the adversary can make the leader faulty with probability $\geq \frac{1}{2}+\varepsilon$. That means that the adversary implements a blocking algorithm $\mathcal{BL}$ which, given as inputs n encrypted pairs

$$\left(E^1(c_1), E^1(l_1)\right), \ldots, \left(E^n(c_n), E^n(l_n)\right)$$

(where $n-1$ of the l_i's are 0, and their corresponding c_i's are random bits, exactly one l_i is 1, and the c_i corresponding to this i is 1), outputs $n-t$ pairs, that contain the $(E(1), E(1))$ with probability no greater than $\frac{1}{2}-\varepsilon$. (The messages that $\mathcal{BL}$ outputs correspond to the correct processors, while the blocked ones are those originating in processors that are made faulty.)

We build a *distinguisher* for the encryption function E. The distinguisher is a polynomial time algorithm that 'behaves' differently when given as input the pair $(E(1), E(1))$ versus the pair $(E(\text{RANDOM BIT}), E(0))$; (RANDOM BIT $\in \{0,1\}$ is randomly chosen). To this end, we first create $n-1$ pairs of probabilistic encryptions

$$\left(E^1(\text{RANDOM BIT}_1), E^1(0)\right), \ldots$$

$$\ldots, \left(E^{n-1}(\text{RANDOM BIT}_{n-1}), E^{n-1}(0)\right)$$

(where $\text{RANDOM BIT}_i \in \{0,1\}$ is randomly chosen).

Given a pair $(E(x), E(y))$, the $n-1$ pairs are joined to it and we feed the n pairs to $\mathcal{BL}$ (in a random order). If $x=1, y=1$, then this is a random instance of the event "exactly one leader volunteered and its random bit is 1". Therefore, according to the assumption, $\mathcal{BL}$ will output $(E(x), E(y))$ with probability no greater than $\frac{1}{2}-\varepsilon$. If, on the other hand, $x = \text{RANDOM BIT}, y = 0$, then the inputs to $\mathcal{BL}$ are n pairs whose elements are probabilistic encryptions of identical sources. Hence the outputs are just a random subset of $n-t$ out of these n encryptions, and so the original pair $(E(x), E(y))$ is output with probability at least $\frac{n-t}{n} \geq \frac{1}{2}$.

The net effect of the whole procedure is that if $x=1, y=1$ then $(E(x), E(y))$ is output with probability $\leq \frac{1}{2}-\varepsilon$, while if $x = \text{RANDOM BIT}, y = 0$ then $(E(x), E(y))$ is output with probability $\geq \frac{1}{2}$. Thus a distinguisher for E is constructed using the adversary's $\mathcal{BL}$. □

As we remarked earlier, the problem of key distribution can be solved by having each processor p broadcast its own (individually generated) public key E_p. In the dynamic-broadcast model, processors spend an extra initial round broadcasting their public keys. This can be done once, during system initialization, or anew with every coin toss execution. This guarantees that there are $n-t$ processors whose public keys are known to everyone. During a coin toss broadcast, each processor encrypts messages with the public key of the recipient, or sends nothing if the recipient's public key is not known. A different set of t processors may fail during the toss than fail during the key distribution. By using $2t$ as the maximum number of faults possible, the proof of Theorem 3 shows that COIN TOSS produces a weakly global coin in the dynamic-broadcast model for $n > 4t$.

In the dynamic-reception case, processors participate in two rounds of broadcasting and forwarding public keys. Once again, this can be done once, during system initialization, or anew with every COIN TOSS execution. Let $n = \alpha t$. In the full paper, we prove that a two-round exchange is sufficient to guarantee that all but at most $(\alpha/\alpha - 1)t$ public keys have been fully distributed. Conversely, there is a set of at least $n - (\alpha/\alpha - 1)t$ processors whose public keys are known by every processor. Setting $T = t + (\alpha/\alpha - 1)t$ as the new bound on the number of faults, the proof of Theorem 3 shows that COIN TOSS produces a weakly global coin in the dynamic-reception model for $n > 4T$.

5. The Asynchronous Case

In this section we abandon the assumption that processors run in synchronous rounds. Correct processors may run arbitrarily fast or slow, and messages may arrive out of order, or take arbitrarily long to arrive, even in the absence of failures. We make the following assumption about the nature of failures in the asynchronous model.

Asynchronous Failures: Except for a set of at most t sending processors, all messages sent by every processor are eventually delivered.

If m processors send to processor p, this implies that p eventually receives at least $m - t$ of those messages.

We assume that processors begin each consensus protocol with the same value in its local round counter. In our algorithms, processors append the current value of the round counter to each message. Each processor counts local rounds, consisting of a broadcasting phase and a reception phase. During the reception, the processor waits for exactly $n - t$ messages with the current round number (some of which may already be received, and stored locally). (For simplicity, we assume that extra messages with a given round number are discarded.) In general, it would be foolish for any processor to wait for more than $n-t$ messages from a given round, since failures may prevent more than this many messages from ever arriving.

We consider two failure models for the asynchronous case, the asynchronous message-oblivious and asynchronous message-dependent models. These both assume the asynchronous failure assumption, adding, respectively, the message-oblivious and message-dependent limitations from the synchronous case. In these models, the adversary has full control of the order and timing of arriving messages and of the rates of internal clocks, and is therefore more powerful than in the synchronous case. The adversary is limited in only two ways. The constraints of the failure assumption require it to eventually deliver enough messages, and the message-oblivious and message-dependent limitations restrict the information it may use to determine its strategy. Despite the adversary's increased power, a suitable modification will still guarantee that agreement is reached in constant expected time for $n \geq 6t$.

Before we describe the modification, let us first explain why it is needed at all. One might be tempted to argue "*once the coin tosses are hidden (by assumption or by encryption), the adversary cannot know which messages to block and so everything works just as it did in the synchronous case*". This naive argument is incorrect because an adversary can in general infer information about messages from the way that processors who receive these messages react to them. If the reaction of each processor to $n - t$ coin toss messages is sufficient to infer that a single processor volunteered, the adversary can successively deliver different subsets of messages to different processors, implementing a simple elimination procedure to determine the identity of the leader. The leader's messages can then be held back from the remaining processers until they have finished the coin toss, rendering the leader useless.

Our solution is to add an acknowledgement round to the end of the coin toss, so that no processor will exit the coin toss routine until receiving acknowledgements from $n-t$ processors. This means that during a coin toss, no processor's observable behavior depends upon the values of the coin toss messages, until at least $n-t$ processors have each received $n - t$ current coin toss messages. The choice of messages to deliver to these $n - t$ processors is thus independent of their contents. The parameter t is chosen so that by this time there is a constant probability that it is too late for the adversary to effect the outcome.

Because of the round structure we impose, the leader's messages are only effective if they are among the *first* $n-t$ messages for that round to arrive at $\lfloor n/2 \rfloor + t + 1$ other processors. For the asynchronous case this will be our definition of a *persuasive* processor for a given round. Our algorithms work by guaranteeing a constant probability that a single volunteer will be persuasive. Without making it explicit in the code, we implicitly assume that a round counter is locally maintained and incremented by each processor. When we say that a processor recieves $n - t$ messages, we mean that it reads messages from its buffer until recieving $n - t$ messages with its current round number. The code for the asynchronous, message-oblivious model is as follows.

Code for processor P:

1. **function** ASYNCHRONOUS COIN TOSS:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast (c_p, l_p)
5. receive the first $n-t$ (c,l) messages with current round number
6. broadcast an acknowledgement
7. receive $n-t$ acknowledgements with current round number
8. **if** a unique message with $l=1$ was received
9. **then** COIN TOSS $\leftarrow c$ of that message
10. **else** COIN TOSS $\leftarrow 0$

Theorem 4: *The function* ASYNCHRONOUS COIN TOSS *produces a weakly global coin in the asynchronous, message-oblivious fault models, provided* $n \geq 6t$.

Proof: From the discussion above, the adversary's choice of the initial $n-t$ coin toss messages to deliver to some set of $n-t$ processors is made independently of the message's contents. We will pick t so that delivering $n-t$ messages to $n-t$ processors implies that some constant fraction α of the processors are persuasive. Since the adversary cannot identify the single volunteer, the leader will have probability α of being persuasive. (This is very similar to the argument for the dynamic-broadcast, synchronous case). Let αn be the number of persuasive processors. There are $(n-t)(n-t)$ messages delivered, of which at most αn^2 come from persuasive processors, and $(n-\alpha n)(\lfloor n/2 \rfloor + t)$ from others. From $(n-t)^2 \leq \alpha n^2 + (n-\alpha n)(\lfloor n/2 \rfloor + t)$, we derive $\frac{n\lceil n/2 \rceil - 3t + t^2}{\lceil n/2 \rceil - t} \leq \alpha$. The number of faults, t, must be such that α is forced by this relation to be a positive fraction. This occurs when $t < (3-\sqrt{7})/2$. In particular, when $t \leq n/6$, α is about $1/12$. □

To defeat a message-dependent adversary in the asynchronous case, we make the same alteration as in the synchronous case, encrypting the random bits.

Code for processor P:

1. **function** ASYNCHRONOUS COIN TOSS:
2. $l_p \leftarrow$ LEADER
3. $c_p \leftarrow$ RANDOM BIT
4. broadcast $(E(c_p), E(l_p))$
5. receive and decrypt the first $n-t$ $(E(c), E(l))$ messages with current round number
6. broadcast an acknowledgement
7. receive $n-t$ acknowledgements with current round number
8. **if** recieved a unique message with $l=1$
9. **then** COIN TOSS $\leftarrow c$ of that message
10. **else** COIN TOSS $\leftarrow 0$

Theorem 5: *Under the assumption* $(*)$, *the modified function* ASYNCHRONOUS COIN TOSS *produces a weakly global coin in the asynchronous message-dependent model, provided* $n \geq 6t$.

Proof Sketch: As in Theorem 3, we will argue that an adversary who can prevent a successful coin toss is capable of inverting the cryptosystem. To prevent a successful coin toss, the adversary must be capable of preventing a temporary leader from becoming persuasive. As we argued in the proof of Theorem 4, the acknowledgement round prevents any processor from leaking information about a particular coin toss until at least $n-t$ processors each receive $n-t$ encrypted coin toss messages. If $n \geq 6t$, the adversary must use the contents of these encrypted messages to prevent a temporary leader from becoming persuasive. If he succeeds substantially more often than he would by guessing at random, he could use this capability to invert the cryptosystem. □

As in the synchronous case, it is possible to perform key distribution as part of the protocol, at some cost in the number of faults tolerated. Full details will appear in the final paper.

6. Using A Weakly Global Coin In Achieving Consensus

In this section we present an agreement algorithm which can be implemented using a weakly global coin. For simplicity of presentation, the algorithm given here is binary (reaching agreement on one bit), and is basically a modification of those in [Be] and [BT]. It can easily be extended to be multivalued (reaching agreement on arbitrary values) using the technique of Turpin and Coan [TC].

First, we give an informal description of the algorithm, and then we give a formal description. The algorithm is organized as a series of epochs of message exchange. Each epoch consists of several rounds. The round structure is provided automatically in the synchronous models. In the asynchronous models, the round structure is imposed locally by each processor, as was discussed earlier. In this case, reaching consensus in 'constant expected time', means that each processor will complete the protocol within a constant expected number of local rounds.

We describe the algorithm for the processor P. (All processors run the same code.) Epoch and round numbers are always the first two components of each message. The variable CURRENT holds the value that processor P currently favors as the answer of the agreement algorithm. At the start of the algorithm CURRENT is set to processor P's input value. In the first round of each epoch, processor P broadcasts CURRENT. Based on the round 1 messages received, processor P changes CURRENT. If it sees at least $\lfloor n/2 \rfloor + 1$ round 1 messages for some particular value, then it assigns that value to CURRENT; otherwise, it assigns the distinguished value "?" to CURRENT. In the second round of each epoch, processor P broadcasts the new CURRENT. Next, the COIN TOSS subroutine is run. Based on the round 2 messages received, processor P either changes CURRENT again, or decides on an answer and exits the algorithm. Let ANS be the most frequent value (other than "?") in round 2 messages received by P. Let NUM be the number of such messages. There are three cases depending on the value of NUM. If NUM $\geq \lfloor n/2 \rfloor + 1$ then processor P decides on the value ANS and exits the algorithm. If $\lfloor n/2 \rfloor \geq$ NUM ≥ 1 then processor P assigns the value ANS to the variable CURRENT and continues the algorithm. If NUM $= 0$ then processor P assigns the result of the coin toss to the variable CURRENT.

Code for processor P

1. **procedure** AGREEMENT(INPUT):
2. CURRENT $\leftarrow$ INPUT
3. **for** $e \leftarrow 1$ **to** ∞ **do**
4. broadcast $(e, 1,$ CURRENT$)$
5. receive $(e, 1, *)$ messages
6. **if** for some v there are $\geq \lfloor n/2 \rfloor + 1$ messages $(e, 1, v)$
7. **then** CURRENT $\leftarrow v$
8. **else** CURRENT $\leftarrow$ "?"
9. broadcast $(e, 2,$ CURRENT$)$
10. receive $(e, 2, *)$ messages
11. ANS $\leftarrow$ the value $v \neq$ "?" such that $(e, 2, v)$ messages are most frequent
12. NUM $\leftarrow$ number of occurrences of $(e, 2,$ ANS$)$ messages
13. COIN $\leftarrow$COIN TOSS
14. **if** NUM $\geq \lfloor n/2 \rfloor + 1$ **then** decide ANS, broadcast ANS DECIDED and **terminate**
15. **elseif** NUM ≥ 1 **then** CURRENT $\leftarrow$ ANS
16. **else** CURRENT $\leftarrow$COIN

We make several remarks about the algorithm. COIN TOSS, depending on the fault model, is one of the protocols described earlier for producing a weakly global coin. In message descriptions, "*" is a wild-card character that matches anything. The **terminate** statement in step 14 hides a model-dependent detail. In the synchronous models, processors simply halt. In the asynchronous models, processors send four more rounds of messages, as though they were executing the for-loop a final time, but without bothering to recieve any messages. This is needed because, once the first correct processor decides and terminates, the other correct processors may not decide until the next epoch (as we argue below). The extra broadcasts are solely to insure that these 'tardy' processors receive a sufficient number of messages during each round of that epoch. (Recall that in the asynchronous fault models, processors must wait for $n - t$ messages during each reception).

Define *value* as a legal input to the algorithm, either 0 or 1. Specifically, "?" is not a value.

The following claims establish the desired properties of the agreement algorithm.

Lemma 6: *During each epoch, conflicting values are never sent in round 2 (step 9).*

Theorem 7 establishes that our algorithm never produces a wrong answer and that in each epoch there is at least one coin toss value that will terminate the algorithm.

Theorem 7: *The algorithm has the following three properties.*

Validity: If value v is distributed as input to all processors, then all correct processors decide v during epoch 1.

Agreement: Let e be the first epoch in which a correct processor decides. If correct processor P decides v in epoch e, then by the end of epoch $e+1$ all correct processors decide v.

Termination: In any epoch, e, there is at least one value which, if it is adopted by $\lfloor n/2 \rfloor + t + 1$ processors executing the assignment in step 16, will cause all correct processors to decide by the end of epoch $e+1$.

The termination property guarantees that a weakly global coin will lead to a decision with constant probability. The agreement property guarantees that once a single processor decided, all others will decide in the next epoch, regardless of the adversary behaviour. In particular, this holds for the asynchronous, message-dependent model, the one in which the adversary has the most power. The proofs follow by the techniques of [Be] and [CC]. We omit details here.

7. Lower Bounds

In this section we show that our upper bound is almost optimal in a strong sense. We demonstrate a lower bound on the tail of the distribution of non-termination probabilities for any randomized agreement algorithm. This lower bound holds for the case of a non-adaptive adversary in the fail-stop model, and therefore in the stronger failure models as well.

Let $\mathcal{A}$ be a randomized agreement algorithm that is resilient to t processor failures and never errs. Such an algorithm, together with the n input values and n (possibly infinite) $0-1$ strings (outcome of individual coin tosses) totally determine the behavior of each processor. Denote by q_k the maximum probability, over all (non-adaptive) adversarial strategies, that $\mathcal{A}$ does not terminate in k rounds ($k \leq t$).

Theorem 8: $q_k \geq \frac{1}{2}\left(2\lceil \frac{n}{\lfloor t/k \rfloor} \rceil\right)^{-k}$.

Proof: Following the Dolev and Strong lower bound proof for deterministic agreement algorithms, we construct a chain $S_1, S_2, \ldots, S_m$ of partially specified executions in the fail-stop model. Each S_i consists of k rounds, and it specifies the identity of faulty processors, their failure time, and identity of receivers of their last round message. (Thus every S_i can be viewed as a strategy of a non-adaptive adversary). These partially specified executions include the initial input values to each processor. Together with the n coin tossing strings $\vec{c} = (c_1, c_2, \ldots, c_n)$, each S_i gives a complete specification of an execution, which we'll denote by $S_i^{\vec{c}}$. For every $\vec{c}$, the following properties hold:

1) $S_i^{\vec{c}} \sim S_{i+1}^{\vec{c}}$ (i.e. both executions look the same for at least one processor which is non-faulty through the k-th round).

2) If all processors agree in $S_1^{\vec{c}}$, then they must agree on the value 1.

3) If all processors agree in $S_m^{\vec{c}}$, then they must agree on the value 0.

4) $m \leq 2\left(2\lceil \frac{n}{\lfloor t/k \rfloor} \rceil\right)^k$

We show that $q_k \geq 1/m$. Substituting (4), this establishes the result.

Assume to the contrary that $q_k < 1/m$. Then the probability (over all $\vec{c}$'s) that $\mathcal{A}$ *does not* terminate in $S_1^{\vec{c}}$ or in $S_2^{\vec{c}}$ ... or in $S_m^{\vec{c}}$ is at most $m \cdot q_k < 1$. Hence the set of $\vec{c}$'s for which $\mathcal{A}$ terminates in all $S_i^{\vec{c}}$ $(1 \leq i \leq m)$ has measure > 0. For each $\vec{c}$ in this set, all correct processors will decide on the value 1 in $S_1^{\vec{c}}$ (by property 2). Hence, by property 1, there is a correct processor which will decide on the value 1 in $S_2^{\vec{c}}$, and therefore, by the agreement requirement, all correct processors will decide on the value 1 in $S_2^{\vec{c}}$. Carrying this argument inductively, it follows that for all $1 \leq i \leq m$, all correct processors will decide on the value 1 in $S_i^{\vec{c}}$. But for $i = m$, this contradicts property 3. □

Acknowledgment

We would like to thank Brian Coan for his comments on an earlier version of this manuscript.

References

[ABCGM] B. Awerbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali, "How to Implement Bracha's $O(\log n)$ Byzantine Agreement Algorithm", unpublished manuscript, MIT.

[ACGS] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr, "RSA/Rabin Bits Are $\frac{1}{2} + \frac{1}{poly(\log N)}$ Secure", *Proc.* 25^{th} *Annual Symposium on Foundations of Computer Science* (1984), pp. 449–457.

[Be] M. Ben-Or, "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols", *Proc.* 2^{nd} *Annual ACM Symposium on Principles of Distributed Computing* (1983), pp. 27–30.

[Br] G. Bracha, "An $O(\lg n)$ Expected Rounds Randomized Byzantine Generals Algorithm", *Proc.* 17^{th} *Annual ACM Symposium on Theory of Computing* (1985), pp. 316–326.

[BT] G. Bracha and S. Toueg, "Resilient Consensus Protocols", *Proc.* 2^{nd} *Annual ACM Symposium on Principles of Distributed Computing* (1983).

[CC] B. Chor and B. Coan, "A Simple and Efficient Randomized Byzantine Agreement Algorithm", *Proc.* 4^{th} *Symposium on Reliability in Distributed Software and Database Systems* (1984), pp. 98-106. To appear in *IEEE Transactions on Software Engineering.*

[DS] D. Dolev and H. R. Strong, "Polynomial Algorithms for Multiple Processor Agreement", *Proc.* 14^{th} *Annual ACM Symposium on Theory of Computing* (1982), pp. 401–407.

[FLP] M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", *Proc.* 2^{nd} *ACM Symposium on Database Systems* (1983), pp. 1–7.

[GM] S. Goldwasser and S. Micali, "Probabilistic Encryption", *Jour. of Computer and System Sciences*, 28(2) (1984), pp. 270-299.

[GMR] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof", *Proc.* 17^{th} *Annual ACM Symposium on Theory of Computing* (1985), pp. 291–304.

[KY] A.R. Karlin and A.C. Yao, "Probabilistic Lower Bounds for Byzantine Agreement and Clock Synchronization", unpublished manuscript, Stanford University.

[LSP] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems* 4(3) (1982), pp. 382–401.

[PSL] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults" *JACM* 27(2) (1980) pp. 228-234.

[Pi] S. Pinter, *Distributed Computation Systems*, Ph.D. thesis, Boston University, (1983).

[Ra] M.O. Rabin, "Randomized Byzantine Generals", *Proc.* 24^{th} *Annual Symposium on Foundations of Computer Science* (1983), pp. 403–409.

[Ro] E. C. Rosen, "Vulnerability of Network Control Protocols: An Example" *ACM SIGSOFT Software Engineering Notes*, 6(1) (1981), pp. 6–8.

[TC] R. Turpin and B. Coan, "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement", *Information Processing Letters* 18(2) (1984), pp. 73–76.

[Y1] A.C. Yao, "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symposium on Foundations of Computer Science*, (1982), pp. 80-91.

[Y2] A.C. Yao, private communication.