Robust and Speculative Byzantine Randomized Consensus with Constant Time Complexity in Normal Conditions

Bruno Vavala*† and Nuno Neves*
*LaSIGE, University of Lisbon, Portugal
†Carnegie Mellon University, U.S.
Email: {vavala, nuno}@di.fc.ul.pt

***Abstract*—Randomized Byzantine Consensus can be an interesting building block in the implementation of asynchronous distributed systems. Despite its exponential worst-case complexity, which would make it less appealing in practice, a few experimental works have argued quite the opposite. To bridge the gap between theory and practice, we analyze a well-known state-of-the-art algorithm in normal system conditions, in which crash failures may occur but no malicious attacks, proving that it is fast on average. We then leverage our analysis to improve its best-case complexity from three to two phases, by reducing the communication operations through speculative executions. Our findings are confirmed through an experimental validation.**

***Keywords*-randomized consensus; message passing; normal situations; asynchronous model; speculative algorithm**

I. Introduction

Is randomized consensus practical? Researchers have been asking this question for the past few decades and yet they cannot reach an unanimous conclusion. On the one hand, theoreticians believe it is inefficient due to its exponential worst-case complexity. On the other hand, practicians believe that many theoretical models are too extreme to describe the real world, and eventually decide to integrate in their systems well-performing algorithms, though theoretically inefficient in the worst-case. In this paper we reconcile these positions to some extent by providing a theoretical analysis of a randomized consensus algorithm in a more practical model, eventually proving that it is fast.

Bracha's algorithm [5] has attracted a lot of attention because of its pivotal contribution to the area and for its simplicity. Today it still represents the state-of-the art in Byzantine fault-tolerance, in terms of reliability and resiliency. Improving on Ben-Or's result [4], it is the first algorithm able to tolerate the optimal bound of $f = \lfloor \frac{n-1}{3} \rfloor$ malicious failures [14] in a completely asynchronous system by leveraging randomization, thus eluding the impossibility result for deterministic algorithms [11], and without resorting to public key cryptography. However, it is quite inefficient in theory because it takes an exponential number of rounds to terminate. The high complexity is related to the worst-case scenario, in which up to f corrupted processes and an adversarial scheduler having complete knowledge of the system (i.e., processes' internal states, exchanged messages, etc.) aim at subverting it.

Although it is reasonable (and realistic) to assume that the system does not always run in the worst-case scenario, it is not wise to relax completely this assumption. First, the worst-case clearly represents an abstraction of an attack the system can suffer. Second, even if no attack is performed, a bad execution run might simply happen. However, once established that the distributed system is usually reliable, then attacks and failures can only interfere with its efficiency, thereby being detectable as they slow it down and reduce its throughput. These issues can be considered transient because either they can be solved through human intervention, or it is unlikely that they are accidentally long-lasting.

The previous argument would not be useful if the implemented algorithms were slow in all scenarios, simply because normal and abnormal situations would be indistinguishable. In the first case, as the algorithm would keep the same performance even in presence of good conditions, it would raise no concern. The existence of fast algorithms in the worst-case is still an open question but there is a wide belief that it might be difficult to devise one.

In this paper, we prove that Bracha's algorithm terminates in constant time in normal conditions. The result is close to the lower bound given by Attiya and Censor in [3]. Also, building on it, we present a new extension to the Bracha's algorithm so that it converges more rapidly, in terms of communication steps, while retaining the robustness against colluding Byzantine processes. In particular, it uses a speculative execution to reduce the number of phases from three to two, with no additional computational cost in the worst-case. Our experimental results ultimately give further and clearer confirmation that randomized consensus is indeed practical, even more attractive by using speculation.

II. Related Work

As consensus cannot be attained by any deterministic algorithm [11], two main techniques can be followed to circumvent this limitation: relaxing the model, typically assuming *eventual synchrony* (directly in the system model or hiding it in a failure detector), or making the algorithm non-deterministic, typically leveraging randomization. While the first trades correctness with a weaker model, the second instead maintains correctness in the strongest model, though its performance is yet to be better understood and improved. Our work makes a contribution in this direction.

In terms of resiliency, Bracha's algorithm, as well as the speculative one that we will describe, compares favorably against several other works [4], [8] that employ up to $2f$ more processes, though at the price of more expensive communication primitives (i.e., *reliable broadcast*). Firstly, it is widely believed that keeping the number of processes low represents the main practical challenge due to its cost. For several consensus applications more processes mean more machines that need to run diverse software to avoid common vulnerabilities, which could lead to the complete compromise of the system. Secondly, the impact of sending (moderately) more messages is typically noticeable when the underlying hardware is limited or when they have to travel over a wide area. Hence for several deployments (e.g., a LAN with high speed routers) it is possible to maintain a sustainable workload.

In [6], [2] it was shown that by assuming randomness in the environment rather than within the algorithm is sufficient to achieve consensus efficiently (i.e., deterministic algorithms can be proved correct and fast). The drawback of those protocols is that they cannot withstand actions that fall in the scope of the strong adversary model, although in this case only liveness is compromised.

Randomized consensus algorithms under the more realistic message-oblivious model were first considered in [8]. In this model, either no malicious entity acts, or it acts *blindly*, independently from the message content. Also in this case, the algorithms are not resilient-optimal, or cannot withstand malicious failures. The work in [3] extends the previous one and provides a new lower bound on the probability that an algorithm does not terminate after k rounds, which is at least $\lceil \frac{n}{f} \rceil^{-k}$. In our analysis we show that Bracha's algorithm provides a close (up to a constant factor) upper bound.

A few works [7], [1] addressed successfully the problem of Byzantine failures when the adversary only controls the corrupted processes — he does not look at the internal state of the correct processes nor at the messages they exchange with each other. Contrarily to [5], they all prove that consensus can be achieved fast under that model, in polynomial [1] and constant [7] time, respectively. The main limitation of these protocols is that, beyond the fact that they cannot withstand a strong adversary, the low time complexity is attained at the expense of an extremely high number of message exchanges, making their implementation hardly feasible. However, in presence of a weaker adversary and of crash-only failures, it is possible to improve it through gossip algorithms [12].

Finally, we notice that the literature is somewhat poor of experimental evaluation of randomized consensus algorithms, probably because simple algorithms are theoretically inefficient, while more specific ones are rather complex to implement and/or not viable due to the number of processes. In [19] it is investigated the practical impact in a LAN of the impossibility result [11] in a replicated system under high load, ultimately observing that it maintains a positive throughput even in adverse network conditions. In [17], Bracha's algorithm is evaluated in a LAN, and the results show that it takes at most a couple of rounds to terminate. In [16] the same algorithm is instead evaluated in a wireless network, showing that the latency increases a lot with the number of processes. However, as suggested in [20], this is probably due to the high contention among the processes in accessing the wireless medium. Our work provides: theoretical fundaments that sustain and predict these experimental results; and a methodology for the analysis of randomized consensus algorithms in practical scenarios.

III. The Consensus Problem and its Solution

The *Randomized Binary Consensus* problem is defined by 3 properties [10]. The first two are *safety* properties, which are common to most solutions present in the literature: *validity*, if all correct processes start the computation with the same proposal v, this is the only value that can be decided on; *agreement*, no two correct processes decide differently. The third one is the *liveness* property, which specifies that the system must eventually make progress in its computation, eventually taking a decision: *termination*, all correct processes eventually decide with probability 1. This property is the one that usually differentiate *randomized* from *deterministic* consensus, since the second one always ensures decision.

Bracha's Solution. Algorithm 1 presents the main structure of the algorithm [5]. We voluntarily overlooked two primitives of the original algorithm (and so we dubbed it *weak*), which are crucial to withstand Byzantine failures under the strong adversary model: the reliable broadcast primitive (see Section VI) and the message validation procedure. The reason is that our analysis is for the normal conditions where no malicious entity exists, thereby making them unnecessary. In turn, even though we use this simplification, our result still holds for the full algorithm.

The algorithm is composed of three phases. In each one, a process performs a broadcast, waits to receive at least $n-f$ messages, and then makes some computation. Specifically, in the first phase the processes run a sort of pre-agreement, in which a process sets its proposal to the one present in the majority of the received messages. In the second phase instead, a value is set only if the majority of processes proposed it, otherwise a process opts for a default value $\perp$. Such a majority implies (by reasoning about intersecting quorums) that if one value is ever set, then that is the only one any other process can ever choose (beyond the default value). In the third phase, if the processes notice that enough of them have set a particular value, then they safely decide on it, otherwise they toss a coin and start over.

```
$r = 1$                                                  # round number
$\phi$                                                   # phase number
$v = initial\ proposal$
while true do
    $\phi = 1$                                           # 1st Phase
    broadcast($<r, \phi, v>$)
    $\mathcal{M}_r^\phi$ = receive $n - f$ $<r, \phi, *>$-messages
    if $\exists w \in \{0, 1\}, |\{m \in \mathcal{M}_r^\phi, m =<r, \phi, w>\}| > f$ then
        $v = w$
    end if
    $\phi = 2$                                           # 2nd Phase
    broadcast($<r, \phi, v>$)
    $\mathcal{M}_r^\phi$ = receive $n - f$ $<r, \phi, *>$-messages
    if $\exists w \in \{0, 1\}, |\{m \in \mathcal{M}_r^\phi, m =<r, \phi, w>\}| > \frac{n}{2}$ then
        $v = w$
    else
        $v = \perp$                                      # default value
    end if
    $\phi = 3$                                           # 3rd Phase
    broadcast($<r, \phi, v>$)
    $\mathcal{M}_r^\phi$ = receive $n - f$ $<r, \phi, *>$-messages
    if $\exists w \in \{0, 1\}, |\{m \in \mathcal{M}_r^\phi, m =<r, \phi, w>\}| > 2f$ then
        $v = v_d = w$                                    # decision
    else if $\exists w \in \{0, 1\}, |\{m \in \mathcal{M}_r^\phi, m =<r, \phi, w>\}| > f$ then
        $v = w$
    else
        $v$ = coin()                                     # toss a coin
    end if
    $r = r + 1$                                          # next round
end while
```

Algorithm 1: Weak Bracha's Algorithm $f = \lfloor \frac{n-1}{3} \rfloor$

IV. Normal Conditions

We consider a system of n processes, f of which might experience failures (we will define this better later on). The computation proceeds in asynchronous rounds. Rounds are structured in phases, each one composed by a communication operation and some computation. The asynchrony that characterizes the system implies that no assumption is taken on the relative speed of both the processes and the message delivery. The channels are reliable, so a message broadcast eventually reaches all recipients, or a subset of them in case the sender is faulty and crashes. Also, it is assumed that a process can correctly identify the sender of a message, so to avoid impersonation attacks. The computation is said to be *message-driven* to indicate that processes take steps based only on the information that they receive. Progress occurs with the reception of $n - f$ distinct messages for the same round and phase, but processes do not get necessarily the same set of messages.

The Strong Adversary Model. The definition of the powers that an hypothetical adversary has to disrupt the successful execution of the protocol gives an abstraction of the environment. In the literature, the *strong adversary model* typically characterizes the worst-case scenario. In this model, a computationally unbounded adversary (i.e., cryptography cannot be used) is able to: access the internal state of the processes, eavesdrop on the communication among them, tune their execution speed, delay and reorder messages, adaptively corrupt up to f processes which, in case of Byzantine failures, collude against the correct ones.

Normal Situations. A model for *normal conditions* should describe what we would expect in practice, during most of the system execution time. Firstly, we can assume that usually there are no Byzantine failures. Secondly, as we noticed in several experiments [17], [16], [19], the presence of an adversarial scheduler does not also occur. For example, it is safe to make no assumption on: which process first acquires the transmission medium; what scheduling algorithm is used at the switch/router to serve the processes' communication requests; how the channel polling is performed at the process, since there are logical/real links that connect it with all the others. It is also hard (though possible) that each process receives in every phase the worst possible set of messages, potentially delaying the decision and the algorithm's termination.

Therefore, we do not lose generality by considering that each message has the same probability of being received, for several reasons: 1) the algorithm is decentralized, in the sense that there is no leader and all messages have the same weight; 2) the identity of the sender of a message does not really matter, what is important is that the messages received by a process are *distinct*; 3) randomization is already present within the algorithm itself and in particular in the value each message carries. It must be noted that this is not in contrast with the asynchrony assumption. Also, the model considers that no malicious entity is acting, and thus the messages are delivered independently from their content and internal status of the processes. Finally, crash failures can occur obliviously (i.e., they may not necessarily slow down the algorithm).

V. Probabilistic Analysis

In this section we provide the complexity analysis of the Bracha's protocol in *normal conditions*. As the correctness is preserved from the stronger model, the algorithm inherits the exponential time complexity as an upper-bound. We will show however that at each round the algorithm has a constant probability to terminate, so an expected constant time complexity.

Preliminaries. In order to derive our result, we employ some of the tools and techniques from [6]. In particular, we make extensive use of approximations, but we avoid using Markov chains for the analysis. We also take advantage of another old approximation provided in [9].

The approximations are asymptotically precise. In particular, for a large enough population of size n and a probability p that an element of the population is considered a success, the number of successes is modeled through the Binomial distribution $\mathcal{B}(n,p)$, and the probability to get less than (or equal to) i successes is approximated using the Normal distribution $\mathcal{N}(np, np(1-p))$ as follows:

$$P\left(\mathcal{B}\left(n,\ p\right) \leq i\right) \approx \Phi\left(\frac{i-np}{\sqrt{np(1-p)}}\right)$$

where $\Phi(x)$ represents the *Standard Normal distribution.*

Moreover, we frequently use the following inequality between the Hypergeometric ($\mathcal{H}$) and the Binomial ($\mathcal{B}$) distribution. Let n be defined as above, k the number of items considered successes, n_s the sample size. For $i \leq \frac{kn_s}{n} = E(\mathcal{H})$, it can be stated that

$$P\left(\mathcal{H}\left(n,\ k,\ n_s\right) \leq i\right) \leq P\left(\mathcal{B}\left(n_s,\ \frac{k}{n}\right) \leq i\right)$$

The inequality is true because, according to the parameters of the distributions, they both have the same average and $\mathcal{H}$'s variance is less or equal to $\mathcal{B}$'s. So $\mathcal{H}$ is more concentrated around the average.

Additionally, we will use the following Chernoff Bound (CB, Theorem 1), which shows that in a stochastic process involving n binary independent random variables, the sum of the variables is concentrated around the average with high probability (w.h.p.).

Theorem 1 (Chernoff Bound [15]): Let $X_1, \ldots, X_n$ be independent poisson trials such that $Pr(X_i) = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then, for $0 < \delta < 1$, $Pr(|X-\mu| \geq \delta\mu) \leq 2e^{-\mu\delta^2/3}$.

Finally, the following theorem describes the normal approximation to a general Binomial distribution.

Theorem 2 (De Moivre-Laplace [9]): Consider a Binomial distribution $\mathcal{B}\left(n,\ p\right)$, with average $\mu = np$ and standard deviation $\sigma = \sqrt{np(1-p)}$. For fixed z_1, z_2,

$$P\left(\mu + z_1\sigma \leq \mathcal{B}\left(n,\ p\right) \leq \mu + z_2\sigma\right) \overset{n\rightarrow\infty}{=} \Phi(z_2) - \Phi(z_1)$$

Overview of the Analysis. We proceed phase by phase, describing the properties of the array of proposals in the system. In particular, we will be focused in understanding under what conditions in the second phase (most of) the processes either set a default value, so they will not reach a decision, or a specific value that is then decided on.

Lemma 1 specifies a well-known property of the original algorithm [5]. Basically it states that the decision of one process at some round implies the consensus termination property. Lemma 2 represents the crucial part of the final theorem. It allows us to understand under what condition processes can set a bad (default value) or a good (either 0 or 1) proposal following the message exchange in the second phase. Surprisingly, we get an "everyone or no one"-style result, asymptotically. Lemma 3 describes under what conditions, by the end of the first phase, the difference between the number of processes that propose the two value is $\mathcal{O}(n)$. Lemma 4 quantifies the initial difference in the number of processes that propose the two values, such that they have a biased constant probability to set the most frequent one at the end of the first phase. Finally, Theorem 3 constitutes our main result. We proceed bottom-up: starting from a good configuration in the last phase that leads the processes to a decision with high probability, we go back to the coin tossing procedure, investigating what conditions must be ensured in order to reach that good final configuration. Eventually, the first condition can be met with constant probability.

Analysis in Normal Conditions.

Lemma 1: If a process p decides at the end of round r, then all processes decide by the end of round $r+1$.

Proof: If p decides on v then it received at least $2f+1$ messages carrying that value in the third phase. Since $n = 3f+1$, all the other processes receive at least $f+1$ messages with v. If they do not decide in round r, they do not toss a coin but rather set v, thereby starting round $r+1$ with the same value. Since the algorithm is correct under the validity property, then all the (remaining) processes decide by the end of the round. ■

Lemma 2: Let $0 < k_2 < \frac{1}{2}$ be a constant. If $\frac{n}{2} + k_2 n$ processes propose v at the beginning of the second phase in a round r, then at the end of the phase with probability 1: if $k_2 < \frac{1}{4}$, all processes set the default value; if $k_2 > \frac{1}{4}$, all processes set v.

Proof: A process p sets v in the second phase if it receives more than $\frac{n}{2}$ messages carrying that value. The probability that this happens follows a hypergeometric distribution that can be computed and bound as follows:

$$P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_2 n,\ n-f\right) > \frac{n}{2}\right) = \\ = 1 - P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_2 n,\ n-f\right) \leq \frac{n}{2}\right)$$

$$P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_2 n,\ n-f\right) \leq \frac{n}{2}\right) \leq \\ \leq P\left(\mathcal{B}\left(n-f,\ \frac{1}{2} + k_2\right) \leq \frac{n}{2}\right) \\ \approx \Phi\left(\frac{\frac{n}{2} - \frac{2n}{3}(\frac{1}{2} + k_2)}{\sqrt{\frac{2n}{3}(\frac{1}{2}+k_2)(\frac{1}{2}-k_2)}}\right) \\ = \Phi\left(\frac{\frac{n}{6} - \frac{2k_2 n}{3}}{\sqrt{\frac{2n}{3}\frac{1-4(k_2)^2}{4}}}\right) = \Phi\left(\frac{\sqrt{\frac{2n}{3}}\left(\frac{1}{4} - k_2\right)}{\sqrt{\frac{1-4(k_2)^2}{4}}}\right)$$

For a large enough n, the probability obtained is $1 - \mathcal{O}(e^{-n})$

for $k_2 < \frac{1}{4}$ and it is $\mathcal{O}(e^{-n})$ for $k_2 > \frac{1}{4}$. So, depending on k_2, either the considered event or its complement will succeed with high probability. Hence, we have (for $k_2 > \frac{1}{4}$)

$$P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_2 n,\ n - f\right) > \frac{n}{2}\right) \geq 1 - \mathcal{O}(e^{-n})$$

$$P\left(\text{each process sets } v\right) \geq \left(1 - \mathcal{O}(e^{-n})\right)^n \overset{n \to \infty}{=} 1$$

The case when $k_2 < \frac{1}{4}$ is symmetrical. ■

Lemma 3: If a process has a constant probability $p > \frac{1}{2}$ to set a value v at the end of the first phase then, for some constant $0 < k_2 < p - \frac{1}{2}$, at least $\frac{n}{2} + k_2 n$ processes set v with high probability at the end of the first phase.

Proof: Without loss of generality let us assume that, for some constant $c > 0$, $p = \frac{1}{2} + c$. Considering all n processes, the number of those that set v follows a Binomial distribution $\mathcal{B}(n, p)$, whose average is $\mu = \frac{n}{2} + cn$. A simple application of the CB (Theorem 1) shows that, for any constant $c' > 0$, $P(\mathcal{B}(n,p) \leq \mu - c'n) \leq e^{-\Theta(n)}$ (because, in Theorem 1, δ would be a constant dependent on c and c'). In particular, this still holds if we take $0 < c' < c$. Now consider a constant $k_2 > 0$ such that $c' + k_2 = c$. The result immediately follows because the complementary event happens with high probability, that is $P(\mathcal{B}(n,p) > \mu - c'n) \geq 1 - e^{-\Theta(n)}$, and $\mu - c'n = \frac{n}{2} + k_2 n$. ■

Lemma 4: Let $k_1 > 0$ be a constant. If $\frac{n}{2} + k_1\sqrt{n}$ processes propose v at the beginning of the first phase in a round r, then there is at least a constant probability, namely $\Phi(1.63k_1) > \frac{1}{2}$, that a process sets v at the end of the first phase.

Proof: A process sets v in the first phase if it received a majority of messages carrying that value. The probability that this happens follows a hypergeometric distribution that can be computed and bound as follows:

$$P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_1\sqrt{n},\ n - f\right) > \frac{n}{3}\right) = \tag{1}$$

$$= 1 - P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_1\sqrt{n},\ n - f\right) \leq \frac{n}{3}\right)$$

$$P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_1\sqrt{n},\ n - f\right) \leq \frac{n}{3}\right) \leq$$

$$\leq P\left(\mathcal{B}\left(n - f,\ \frac{1}{2} + \frac{k_1}{\sqrt{n}}\right) \leq \frac{n}{3}\right)$$

$$\approx \Phi\left(\frac{\frac{n}{3} - \frac{2n}{3}(\frac{1}{2} + \frac{k_1}{\sqrt{n}})}{\sqrt{\frac{2n}{3}(\frac{1}{2} + \frac{k_1}{\sqrt{n}})(\frac{1}{2} - \frac{k}{\sqrt{n}})}}\right)$$

$$= \Phi\left(\frac{-\frac{2k_1\sqrt{n}}{3}}{\sqrt{\frac{2n}{3}\frac{n-4k_1^2}{4n}}}\right) = \Phi\left(-\sqrt{\frac{2k_1^2(4n)}{3(n - 4k_1^2)}}\right)$$

$$\leq \Phi\left(-\sqrt{\frac{8k_1^2}{3}}\right) \qquad = \Phi\left(-1.63k_1\right)$$

Substituting in the equation above and since $k_1 > 0$

$$P\left(\mathcal{H}\left(n,\ \frac{n}{2} + k_1\sqrt{n},\ n - f\right) > \frac{n}{3}\right) \geq 1 - \Phi\left(-1.63k_1\right)$$

$$= \Phi\left(1.63k_1\right) > \tfrac{1}{2}$$ ■

Theorem 3: In normal situations, the Bracha's algorithm terminates at least with constant probability, namely 0.40, hence in expected 2.5 rounds.

Proof: Let us assume that all processes propose v in the third phase of a round r. This case is ensured with probability 1 by Lemma 2 provided that there is a constant $k_2 > \frac{1}{4}$, such that $\frac{n}{2} + k_2 n$ processes started the second phase proposing v. This means that, with the same probability, all processes necessarily take a decision by the end of round r — actually the decision of one correct process in r would be sufficient since it implies the termination of the algorithm, from to Lemma 1.

From Lemma 3, constant k_2 exists if the processes have a constant probability $p > \frac{1}{2}$ of setting v following the message exchange of the first phase. From Lemma 4, provided that there exists a constant k_1 such that $\frac{n}{2} + k_1\sqrt{n}$ processes propose v at the beginning of the first phase, such probability is $p = \Phi(1.63k_1)$. By combining the required bound on k_2 with p, we have $\Phi(1.63k_1) - \frac{1}{2} > k_2 > \frac{1}{4}$, so

$$\Phi\left(1.63k_1\right) > \frac{3}{4} \ \Rightarrow\ 1.63k_1 \geq 0.68 \ \Rightarrow\ k_1 \geq 0.42 \tag{2}$$

Since k_1 is related to the distribution at the beginning of the first phase, it depends on the coin tossing at the end of round $r - 1$. The coin tossing follows a Binomial distribution $\mathcal{B}\left(n,\ \frac{1}{2}\right)$, hence from Theorem 2, we have

$$P\left(\left|\mathcal{B}\left(n,\ p\right) - \mu\right| \geq 2k_1\sigma\right) \overset{n \to \infty}{=} 2\left(1 - \Phi(2k_1)\right) \geq 0.40$$

Therefore, if the algorithm does not terminate (no process decide) in round r, then there is at least a constant probability to get a *good* coin tossing that would let the algorithm terminate in round $r + 1$. Such constant probability implies that the algorithm terminates in expected 2.5 rounds. ■

An Improved Approximation. According to our experimental evaluation (see Section VII), the result obtained is loose and does not describe exactly the behavior of the algorithm. The reason is that there is a loss of precision in the Binomial approximation to the Hypergeometric distribution. Building on our analysis, we enhance it by using a result of Feller [9], [18, p.194], which indicates how to approximate the Hypergeometric distribution with the Normal distribution under some conditions.

Feller shows that when the ratio between the sample and the population size tends to a constant then the Hypergeometric distribution $\mathcal{H}\,(n,\ k,\ n-f)$ can be approximated by the Normal distribution as follows

$$\frac{\binom{k}{i}\binom{n-k}{n-f-i}}{\binom{n}{n-f}} \sim \frac{1}{\sqrt{2\pi}\sigma_{\mathcal{H}}} e^{-\frac{x^2}{2}} \qquad \text{with } \frac{i-\mu_{\mathcal{H}}}{\sigma_{\mathcal{H}}} \to x$$

where $\mu_{\mathcal{H}}$ and $\sigma_{\mathcal{H}}$ represent respectively the average and the standard deviation of $\mathcal{H}$, that is:

$$\mu_{\mathcal{H}} = \frac{(n-f)k}{n} \qquad \sigma_{\mathcal{H}} = \sqrt{\frac{(n-f)k(n-k)f}{n^2(n-1)}}$$

Since in our case $n = 3f+1$, we have $\frac{n-f}{n} \to \frac{2}{3}$ and Feller's approximation holds.

To show the impact of such approximation on our previous analysis, we need to consider again the binomial approximation to the hypergeometric distribution, and in particular its average and standard deviation, namely

$$\mu_{\mathcal{B}} = \frac{(n-f)k}{n} \qquad \sigma_{\mathcal{B}} = \sqrt{\frac{(n-f)k(n-k)}{n^2}}$$

It is now clear that $\mu_{\mathcal{B}} = \mu_{\mathcal{H}}$ and $\sigma_{\mathcal{H}} = \sigma_{\mathcal{B}}\sqrt{\frac{f}{n-1}} = \frac{\sigma_{\mathcal{B}}}{\sqrt{3}}$. By translating Feller's approximation in terms of the Standard Normal distribution, we thus get

$$P(\mathcal{H}(n,\ k,\ n-f) \leq i) \overset{\text{Feller}}{\approx} \Phi\left(\frac{i-\mu_{\mathcal{H}}}{\sigma_{\mathcal{H}}}\right) = \Phi\left(\frac{i-\mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}}\sqrt{3}\right)$$

This means that we can easily get a more precise (approximated) result by adjusting with a constant the input to the cumulative distribution function in Theorem 3. This adjustment has no asymptotical impact when such input is positively (resp. negatively) dependent on n, because in that case the probability is exponentially high (resp. low), so it is negligible. Instead, the impact is noticeable when the input is constant, thereby giving a constant probability. This happens in the analysis of the first phase (Lemma 4 and Equation 2). Therefore we restate the main theorem as follows:

Theorem 4: In normal situations, the Bracha's algorithm terminates at least with constant probability, namely 0.63, hence in expected 1.59 rounds.

Proof: Starting from Equation 2 of Theorem 3 and adjusting equations and results according to Feller's approximation, then

$$\Phi\left(1.63\sqrt{3}k_1\right) > \frac{3}{4} \;\Rightarrow\; 1.63\sqrt{3}k_1 \geq .68 \;\Rightarrow\; k_1 \geq 0.24$$

$$P\left(|\mathcal{B}(n,p) - \mu| \geq 2k_1\sigma\right) \overset{n\to\infty}{=} 2\left(1 - \Phi(2k_1)\right) \geq 0.63$$

Such constant probability implies that the algorithm terminates in expected 1.59 rounds. ■

Crash-Failures are Already Included. In order to slow down the algorithm, our analysis suggests that the best strategy is to balance the proposals of 0 and 1 at the beginning of each round by crashing $\mathcal{O}(\sqrt{n})$ processes. In this way the processes would reach a decision after the crash of f of them, hence in $\mathcal{O}(\sqrt{n})$ rounds. However, in normal conditions, crashes occur obliviously and not selectively, so this problem will not affect the performance in practice.

Our analysis did not consider crash failures because they can be abstracted and thus the result still holds in that case. Let us consider a somewhat stronger model in which the adversary only controls the scheduler. In this case, either he has the power, namely enough messages with each value to schedule in some disrupting way, the postponement of the termination, or he cannot succeed. It is clear that in the first case he does not need to crash any process, while in the second case crashes will not help him to acquire the capability to delay the protocol. Therefore, if crash failures do not help to slow down the algorithm in the stronger model then, by reduction, crashes will not be of any aid in our weaker model either.

Comparison with a Lower Bound. It is interesting to note how tight the result is. Attiya and Censor [3] provide a lower bound on the probability to reach consensus under a weak adversary in presence of crash failures. They proved that: for $n = 3f+1$, the probability that a randomized consensus algorithm does not terminate in $k(n-f)$ *steps* is at least $\lceil \frac{n}{f} \rceil^{-k} \approx (0.25)^k$. Roughly speaking, the definition of *step* corresponds to the execution of one *phase at one process*, in our context. Instead, our result states that the probability that the algorithm does not terminate in r rounds (i.e., $3r$ phases), is at most $(1-0.63)^r = (0.37)^r$. Hence, the two bounds are close up to constant factors. The experimental evaluation will show that ours is rather precise.

VI. The Randomized Consensus Algorithm

This section introduces our speculative algorithm. Overall, the algorithm derives its robustness by being correct in the strong adversary model in presence of $f < n/3$ Byzantine failures. Additionally, we leverage our previous analysis to gain speed in more benign scenarios, by reducing the number of phases from three to two. We start by describing the communication primitives that we use, then we present the consensus algorithm.

Communication Primitives

The broadcast primitive is fundamentally an extension of the Reliable Broadcast algorithm described in [6], which is augmented to guarantee the FIFO property.

Reliable Broadcast. The reliable broadcast algorithm [6] achieves asynchronous Byzantine agreement as specified in [14]. It is composed by 3 message exchanges. In the first step, the sender broadcasts m in an *initial* message. In the second step, the processes that receive this message, rebroadcast it in an *echo* message. In the third step, if a process receives $(n+f)/2$ *echo* messages, it broadcasts m

in a *ready* message. Finally, if a process receives $2f+1$ *ready* messages with m, then it delivers m. The algorithm guarantees the following properties:
P1: if the sender of a message m is correct then all the correct processes eventually deliver m;
P2: if a malicious process sends a message, then either all or none of the correct processes deliver m.

FIFO (Reliable) Broadcast. FIFO Broadcast [13] further achieves the following property:
P3: if a process sends a message m before sending a message m', then no correct process delivers m' before having delivered m.
It can be easily implemented on top of Reliable Broadcast as follows. Each process maintains a sequence number *seq*, that it uses to serialize all of its outgoing messages. Also, a process keeps track, for each other process p, of the sequence number of the last message belonging to p it delivered. A process FIFO-delivers a message m if: 1) m has been reliably delivered; 2) m carries the next expected sequence number with respect to its sender.

One should notice that the cost of FIFO delivery is fundamentally null because it does not involve any further message exchange beyond the reliable broadcast. Furthermore, it is important to note that Byzantine processes cannot arbitrarily break the FIFO property without being detected. As the FIFO broadcast is built on top of reliable broadcast, the lower communication level prevents them from sending conflicting messages with the same sequence number.

Message Stratum. The FIFO broadcast gives the possibility to reason about *message strata*. Let us assume that a process p has just started a new round r of computation. Assume also that all the old messages from the other correct processes have been received. This can be easily achieved due to the FIFO property: late messages from previous rounds are delivered first in r by p and then discarded. Now let $next[\cdot]$ be the array of the next sequence numbers of the messages p expects to receive. For each correct process q, p can associate the $next[q]$-th message to first phase of the round, the $(next[q]+1)$-th to the second, and so on. Hence, we say that the set of $(next[\cdot]+i)$-th messages characterizes a (subset of a) *message stratum* (MS). In particular, a MS can be associated to each phase in each round. In the following, since late messages carrying old rounds are discarded and messages with future rounds are buffered to be delivered later, we deal only with the message strata belonging to the round a process is currently computing.

In Algorithm 1 MSs are trivial because: 1) all processes always execute the same phases; and 2) the round and phase number fields (contained in the messages) altogether characterize a sequence number. However, suppose that a process p could voluntarily miss one phase, then the other processes would get desynchronized, namely: a process q waiting for a message from process p cannot distinguish the case where p skipped the transmission or the message lagged behind due to the asynchrony of the system. The FIFO property helps to solve this problem, because the sequence number of a message exposes the execution flow of its sender, namely what it computed *next*.

```
 1: $m_p$                                              # message received from $p$
 2: $stratum_i[\forall p] \quad i=1,2,3$               # set of validated messages
 3: $s(p) = min\{j|\ stratum_j[p]=\emptyset\}$         # p's lowest empty stratum
 4: $r$                                                # current consensus round
 5: On is-expected($m_p$) :
 6:   $(r_p, \phi_p) = (m_p.r, m_p.\phi)$
 7:   if $(s(p), r_p, \phi_p)$ is $(3, r+1, 1)$
 8:   then SS                                          # Speculation Successful
 9:   if $(s(p),r_p,\phi_p) \in \{(1,r,1),(2,r,2),(2,r,3_{spec}),(3,r,3)\}$
10:   then YES
11:   else NO
12:
13: On is-justified($m_p$) :
14:   in $stratum_{s(p)=1}$ : YES
15:   in $stratum_{s(p)>1}$ :
16:     if $\exists$ a set of $n-f$ messages in $stratum_{s(p)-1}$ that
          justifies that $p$ might have set the value in $m_p$
17:     then YES
18:     else NO
19:
20: On recv-validate($stratum_i$) :
21:   $stratum_{j\geq i}[\forall p] = \emptyset$       # clean current and higher strata
22:   while $|stratum_i[]| < n-f$ messages :
23:     $m_p \leftarrow$ FIFO-deliver                  # message received from $p$
24:     if is-expected($m_p$)=$SS$ & is-justified($m_p$) then
25:       $stratum_3[p] = stratum_2[p]$                # reuse p's old msg
26:       postpone $m_p$'s delivery                    # $m_p$ has round $r+1$
27:     else if is-expected($m_p$) & is-justified($m_p$)
          then $stratum_{s(p)}[p] = m_p$
28:     else postpone $m_p$'s delivery or discard it
29:     end if
30:   end while
```

Algorithm 2: Message Stratum Validation

Description of the Consensus Algorithm

Message Stratum Validation. As processes may have different execution flows and some of them may be malicious, it is important that every process is able to match each received message to the correct MS. In other words, processes need a *message validation* procedure, recv-validate in Algorithm 2, to decide if a message belongs to the correct MS. This occurs if the message is *expected* and *justified*.

A message is *expected* (Algorithm 2, line 5) if the information it carries (i.e., round, phase number) belongs: 1) to the round the receiver is currently computing and the index of the lowest empty stratum cell for the sender (at the receiver) matches the phase number in the message (line 9) — for a speculative message (phase 3_{spec}), the index to be matched is 2 — or, 2) to the first phase of the round that immediately follows the one that the receiver is currently computing, provided that the message sequence number is immediately adjacent to the one of the previous message received from the same sender, which carried the phase number 2 (line 7), and was thereby placed in the second message stratum.

In the consensus algorithm, a process sets its proposal according to the received set of messages. Hence, a message carrying some proposal v is *justified* (Algorithm 2, line 13) if the receiver has a set of (older) messages that could have allowed itself to set that proposal previously. Any proposal at the beginning of the round (i.e., in the first phase) is immediately justified. This cannot create any deadlock in the system when the processes wait to justify some messages, because 1) at least $n-f$ of them are correct and follow the protocol, and 2) Property 2 of the reliable broadcast holds, so the processes cannot send conflicting messages.

Our message stratum validation procedure completely replaces Bracha's message validation.

Consensus Algorithm. The consensus algorithm proceeds in asynchronous rounds (see Algorithm 3). Each round is represented by a full or a speculative execution of the *while* loop (line 5), terminating in line 21 and 41, respectively. Each round is structurally composed by 3 phases plus 1 speculative. The speculative phase is identified as 3_{spec}, and it may allow a process to execute only two phases in one round. Each phase is identified by a computation block (line 6, 15, 30), starting with a communication operation. The speculative phase 3_{spec} is embedded together with phase 2. Each message-receive operation is implemented in the recv-validate primitive, which is useful for the processes to recognize the message strata (line 8, 17, 32). A round is *full* if a process participates in all the 3 strata, while it is *speculative* if a process participates only in the first 2 strata, before starting a new round.

If no process executes the speculative phase, processes follow the same execution flow as in [5]. In the first phase, following the message exchange, a process sets its proposal to the value on the majority of the received messages (line 9). If this majority is larger than $n/2$ (line 10) then it speculates by setting phase 3_{spec}, otherwise it sets phase 2. In the next phase, if a process receives $n-f$ speculative messages (necessarily carrying the same proposal), then it speculated successfully, and therefore decides and proceeds to the next round. Instead, a process that cannot speculate successfully must at least try to decide in the third phase.

Let us analyze how the configuration of $stratum_2$ influences a process proposal in phase 3. If a process receives at least $f+1$ speculative messages (line 24), it may be that some other process has already decided — due to asynchrony the remaining f speculative messages may have been delayed. Clearly, for validity's sake, it must be mandatory for it to follow those speculators, by setting the same proposal for the next phase. On the other hand, if strictly less than $f+1$ speculative messages are received, then the speculation necessarily failed at all processes. In this case, it is trickier to set a proposal, since both values 1 or 0 may appear in $stratum_2$ (but only one can be speculative). From the point of view of a process, the speculators may be more than the f it may be able to see, due to the asynchrony. This means that some other process could have received, for instance, $f+1$ speculative messages and thus set the speculative value. Therefore, a process receiving less than $f+1$ speculative messages can set its proposal to a non-speculative value if:

```
1:  $r = 1$                                   # round number
2:  $\phi = 1$                                # phase number
3:  $v = input$                               # initial proposal
4:  $stratum_i$                               # i-th stratum tag identifier
5:  while 1 :
6:    when $\phi = 1$ :                       # 1st Phase/Stratum
7:      FIFO-broadcast($<r, \phi, v>$)
8:      $\mathcal{S}_1^{(r)}$ = recv-validate($stratum_1$)
9:      $v = w$ s.t. $|\{m \in \mathcal{S}_1^{(r)}, m = <r, \phi, w>\}| > f$
10:     if $|\{m \in \mathcal{S}_1^{(r)}, m = <r, \phi, v>\}| > \frac{n}{2}$ then
11:       $\phi = 3_{spec}$                   # speculative execution
12:     else
13:       $\phi = 2$                          # normal execution
14:     end if
15:   when $\phi = 2$ or $\phi = 3_{spec}$ : # 2nd Phase/Stratum
16:     FIFO-broadcast($<r, \phi, v>$)
17:     $\mathcal{S}_2^{(r)}$ = recv-validate($stratum_2$)
18:     if $|\{m \in \mathcal{S}_2^{(r)}, m = <r, 3_{spec}, w>\}| \geq n-f$ &
        $\phi = 3_{spec}$ then
19:       $v_d = v = w$                       # decision
20:       $\phi = 1$
21:       $r = r + 1$                         # next round
22:     else
23:       $\phi = 3$
24:       if $\exists w, |\{m \in \mathcal{S}_2^{(r)}, m = <r, 3_{spec}, w>\}| > f$
25:         or $(|\{m \in \mathcal{S}_2^{(r)}, m = <r, 2, w>\}| > \frac{n}{2}$ &
                 $|\{m \in \mathcal{S}_2^{(r)}, m = <r, 2, *>\}| \geq n - f)$
26:         or $|\{m \in \mathcal{S}_2^{(r)}, m = <r, *, w>\}| \geq n-f$
27:       then $v = w$
28:       else $v = \perp$
29:     end if
30:   when$\phi = 3$ :                        # 3rd Phase/Stratum
31:     FIFO-broadcast($<r, \phi, v>$)
32:     $\mathcal{S}_3^{(r)}$ = recv-validate($stratum_3$)
33:     if $\exists w, |\{m \in \mathcal{S}_3^{(r)}, m = <r, \phi, w>\}| > 2f$ then
34:       $v_d = v = w$                       # decision
35:     else if $\exists w, |\{m \in \mathcal{S}_3^{(r)}, m = <r, \phi, w>\}| > f$ then
36:       $v = w$
37:     else
38:       $v$ = coin()                        # toss a coin
39:     end if
40:     $\phi = 1$
41:     $r = r + 1$                           # next round
```

Algorithm 3: Speculative Randomized Consensus

1) either no speculative message has been received (line 25), which ensures that at most f processes speculated; or 2) the received speculative messages comply with the non-speculative value to be set (line 26[1]). In the other cases it is safer for a process to set the default value, to avoid having more than one proposal in phase 3, beyond the default value. Hence, at most one proposal is locked.

Finally, a process executes the phase 3 any time it is not able to decide in the speculative phase. This phase resumes the original protocol execution, where a process can safely decide on a (locked) proposal (line 34), provided that it does not receive default values in $stratum_3$. However, $stratum_3$ may contain strictly less than $n - f$ phase-3 messages, if a process decided on a speculative value, and this could potentially lead to a deadlock. As we mentioned before, this problem is prevented by the FIFO property and by the assumption that at least $n-f$ processes are correct. In fact, a process expects that some message from some process q in $stratum_3$ may belong to the subsequent round (Algorithm 2, line 7). In this case, it can recognize that q's speculation was successful (Algorithm 2, line 8) and, therefore: 1) it reuses q's previous message, from $stratum_2$, in the third phase (Algorithm 2, line 25) and 2) it postpones the delivery of the new message, which in fact belongs to the $stratum_1$ of the next round. Once this problem is solved, if a process can decide in a round r, then necessarily: 1) all the others receive at least a majority of messages with that decision value, thereby setting it for the subsequent round (line 35); and 2) as all correct processes start round $r+1$ with the same value, then they all decide by the end of round $r+1$. On the other hand, if none of the previous events happens, then a process tosses a coin (line 38) to escape the impossibility of attaining consensus in deterministic asynchronous systems.

Discussion. The need to maintain the algorithm's correctness in the presence of Byzantine failures has a significant impact on performance because several phases must be run. The speculative phase instead is useful to recognize and speed up the algorithm in normal situations: 1) in the worst-case, the Byzantine processes and an adversarial message scheduler can make the speculation either fail or be avoided, while 2) in normal situations, particularly when most processes start with the same proposal, they are able to reach a decision after 2 communication steps (or phases). In the first case, most importantly, since we treat any speculative message as a particular phase-2 message, this means that speculation failure does not impose any further computational cost.

Improving the complexity in normal situations makes sense as these are expected to occur frequently. For instance, let us consider a state machine replication application. Typically, processes that execute the algorithm receive from a client a service request (using reliable-broadcast if the client can be malicious) for which an agreement problem need to be solved to maintain consistency across replicas. In this case, all correct processes start the algorithm with the same proposal and reach the agreement in one round. With the speculative algorithm, our analysis shows that when $\frac{n}{2}+k_2 n$ processes (from Lemma 2, with $k_2 > \frac{1}{4}$) propose the same value, then the agreement is achieved in 2 phases.

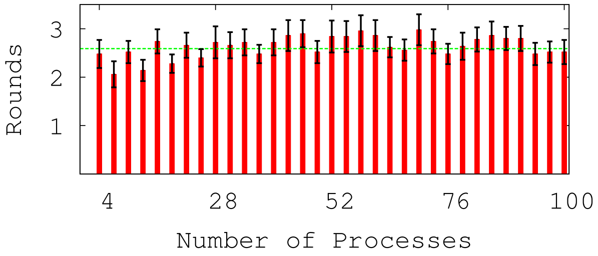

Figure 1: Round complexity of Algorithm 1 in a LAN

VII. Experimental Evaluation

The experiments evaluate the performance (in number of rounds) of Algorithm 1 and 3. They were performed in a cluster of 6 Dell PowerEdge 850 nodes, carrying an Intel Xeon E5520 CPU, 2GB of RAM, and a Broadcom NetXtreme BCM5721 Gigabit Ethernet card. Nodes ran the 2.6.32-21-server Linux kernel and were connected by a Dell PowerConnect 5448 switch. We divided the processes equally among the machines to avoid having faster privileged units. The algorithm was executed to withstand a number of failures ranging in $f = 1, \ldots, 33$. Each process was assigned a unique ID in the range $[0, \ldots, 3f]$. The initial proposal of a process was equal to the parity of its ID, so the initial configuration was divergent. This configuration corresponds to the worst-case scenario that leads to the larger number of executed rounds.

The broadcast primitives were emulated using point-to-point TCP channels. When a process had a message to be transmitted, it randomly picked one of the receivers and then sent the message. This procedure was repeated until the message was transmitted to all destinations. The received operation was carried out in a similar way, the channels were randomly polled to check for available messages. Note that this implementation *does not* change the algorithm's essence.

Figure 1 provides the average number of rounds for termination of Algorithm 1. For each point we present the average of 10 runs and the 95% confidence interval. The results confirm that the algorithm runs in expected constant time. When the processes start to execute, they have divergent proposals and therefore, with very high probability they toss a coin at the end of the first round. At this point our result begins to apply — that is consensus should terminate in 1.59 rounds. Indeed it is possible to confirm that our prediction represents a good approximation (the threshold line is placed at $1 + 1.59$ rounds).

[1] The condition in line 26 is clearly stronger than the one described. Nevertheless it allows for a succinct description while maintaining the algorithm's correctness.

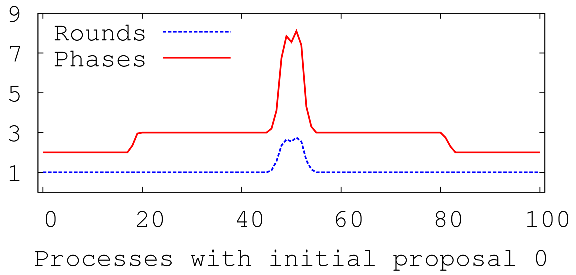

Figure 2: Complexity of Algorithm 3

In relation to Algorithm 3, we performed a simulation with 100 processes, whose results are displayed in Figure 2. We ran the algorithm for all possible initial configurations (by setting the initial proposal of a subset of the processes to 0 and the remaining to 1). In the case of divergent proposals (middle of the graph), the speculative version performs similarly to the original Bracha's algorithm (as it can be confirmed by comparing Figure 1 and 2). For the other cases, our former analysis applies and indeed the speculative algorithm terminates in either 2 or 3 phases. In particular, *almost* precisely as we predicted, it terminates in 2 phases when $n/2 + k_2 n$ processes (with $k_2 > \frac{1}{4}$) start with the same proposal. The reason for such discrepancy lies in the speculative decision condition (line 18), where even just one process that does not speculate may make it false. According to our analysis this event is not likely to happen for a large enough number of processes. However, this problem exists as long as the processes receive only $n - f$ messages, as we simulated. If they are able to base their decisions on larger sets of messages, as it may frequently happen in practice, then it is more likely that the speculation is successful.

VIII. Conclusion

In the paper we show that a well-known, resilient-optimal, Byzantine fault-tolerant (for a strong adversary) algorithm [5] is fast under normal conditions, thus making it attractive for real implementations. Then, building on our analysis we proposed a new speculative algorithm that reduces from three to two the number of phases needed to achieve consensus in the best-case. Our experimental analysis ultimately confirmed our findings.

Acknowledgments

We would like to thank Alysson Bessani for an early review of the work, which helped to improve its presentation.

This work was partially supported by the EC through project FP7-257475 (MASSIF), by the FCT through the Multiannual and the CMU-Portugal Programmes, and the project PTDC/EIA-EIA/113729/2009 (SITAN), and research grant SFRH/BD/51562/2011.

References

[1] I. Abraham, D. Dolev, and J. Y. Halpern. An almost-surely terminating polynomial protocol forasynchronous byzantine agreement with optimal resilience. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 405–414, 2008.

[2] J. Aspnes. Fast deterministic consensus in a noisy environment. *J. of Algorithms*, 45(1):16–39, 2002.

[3] H. Attiya and K. Censor. Lower bounds for randomized consensus under a weak adversary. *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 315–324, 2008.

[4] M. Ben-Or. Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 27–30, 1983.

[5] G. Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the 3rd Symposium on Principles of Distributed Computing (PODC)*, pages 154–162, 1984.

[6] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J.of the ACM*, 32(4):824–840, 1985.

[7] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. of the 25th Symp. on Theory of Computing (STOC)*, pages 42–51, 1993.

[8] B. Chor, M. Merritt, and D. B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *Journal of the ACM*, 36(3):591–614, 1989.

[9] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd Edition*. John Wiley & Sons, 1968.

[10] M. J. Fischer. The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In *Proceedings of the International Conference on Fundamentals of Computation Theory (FCT)*, pages 127–140, Aug. 1983.

[11] M. J. Fischer, N. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2), 1985.

[12] C. Georgiou, S. Gilbert, R. Guerraoui, and D. R. Kowalski. On the complexity of asynchronous gossip. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, page 135, Aug. 2008.

[13] V. Hadzilacos and S. Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical report, Computer Science Department, Cornell University, 1994.

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

[15] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Jan. 2005.

[16] H. Moniz, N. F. Neves, and M. Correia. Turquois: Byzantine consensus in wireless ad hoc networks. In *Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 537–546, 2010.

[17] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Experimental Comparison of Local and Shared Coin Randomized Consensus Protocols. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 235–244, 2006.

[18] W. L. Nicholson. On the Normal Approximation to the Hypergeometric Distribution. *The Annals of Mathematical Statistics*, 27(2):471–483, 1956.

[19] P. Urbán, X. Defago, and A. Schiper. Chasing the FLP impossibility result in a LAN: or, How robust can a fault tolerant server be? In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 190–193, 2001.

[20] B. Vavala, N. F. Neves, H. Moniz, and P. Verissimo. Randomized Consensus in Wireless Environments: a Case Where More is Better. In *Proceedings of the 3rd International Conference on Dependability (DEPEND)*, pages 7–12, 2010.