Byzantine Consensus in the Random Asynchronous Model

George Danezis
Mysten Labs, London, UK
University College London, UK

Jovan Komatovic
EPFL, Lausanne, Switzerland

Lefteris Kokoris-Kogias
Mysten Labs, Athens, Greece

Alberto Sonnino
Mysten Labs, London, UK
University College London, UK

Igor Zablotchi
Mysten Labs, Zurich, Switzerland

Abstract

We propose a novel relaxation of the classic asynchronous network model, called the *random asynchronous model*, which removes adversarial message scheduling while preserving unbounded message delays and Byzantine faults. Instead of an adversary dictating message order, delivery follows a random schedule. We analyze Byzantine consensus at different resilience thresholds ($n = 3f + 1$, $n = 2f + 1$, and $n = f + 2$) and show that our relaxation allows consensus with probabilistic guarantees which are impossible in the standard asynchronous model or even the partially synchronous model. We complement these protocols with corresponding impossibility results, establishing the limits of consensus in the random asynchronous model.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases network model, asynchronous, random scheduler, Byzantine consensus

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.28

Related Version *Full Version*: `https://arxiv.org/abs/2502.09116`

1 Introduction

Byzantine Fault-Tolerant (BFT) consensus is a fundamental primitive in distributed computing, serving as the backbone of many applications, including blockchains [6]. Solving BFT consensus in an asynchronous network model [13] is a well-studied problem [9, 16, 17, 24]. Typically, algorithms in this model assume that the adversary controls (1) a subset of faulty processes and (2) message delays, particularly the *message schedule* – the order in which messages are delivered.

However, asynchronous BFT consensus is constrained by restrictive lower bounds [13, 19]. These lower bounds often stem from the adversary's ability to impose arbitrary message schedules. Requiring an algorithm to function under the worst possible scheduling scenario often renders tasks impossible due to a single, highly unlikely counterexample, that is, a pathological schedule that violates the algorithm's properties [13].

Yet, due to performance considerations, practical implementations of consensus circumvent the limitations of asynchrony by assuming that the schedule is not adversarial. For example, we are aware of asynchronous consensus protocols running in production over commodity

wide-area networks for extensive periods of time without a random coin implementation [11] – so as to avoid the performance overhead of cryptographically-secure randomness – without loss of liveness. This motivates our search for a model to explain this empirical observation.

Another common relaxation that circumvents the limitations of asynchrony is to assume *periods of synchrony*, leading to the widely used *partially synchronous* model [12]. However, protocols based on partial synchrony lose liveness outside of these periods, which may occur due to poor network conditions or denial-of-service (DoS) attacks [15].

In this paper, in order to provide a sound basis for non-adversarial scheduling as assumed by some modern practical systems, and to avoid the limitations of partial synchrony, we propose an alternative relaxation of the asynchronous model: we study asynchronous networks without adversarial scheduling. Specifically, we ask: *What becomes possible in a network where message delays are unbounded, but the message schedule is not adversarial?*

To explore this question, we introduce a new variant of the asynchronous model, which we call the *random asynchronous model*. In this model, message delays remain unbounded, and the adversary still controls a subset of processes. However, the message schedule is no longer adversarial; instead, messages are delivered in a *random* order. Our aim is not to argue that this model is a more realistic description of real networks; rather, we view it as a useful abstraction, inspired by assumptions made in practice [11], that lets us analyze the non-adversarial schedule assumption and explore which tasks it enables under unbounded delays while preserving Byzantine faults. This work offers a new perspective on the role of adversarial scheduling in the standard asynchronous setting: by removing the adversary's scheduling power, we explore what becomes achievable and what remains fundamentally impossible. Our work is a part of the broader line of research that explores relaxations of the fully adversarial scheduler: from the fair-scheduler abstraction of Bracha and Toueg [8], through Aspnes's noisy scheduling for shared memory [4], to recent DAG-based BFT systems such as Tusk [11] and Mahi-Mahi [16].

To isolate the impact of random scheduling, we consider only deterministic algorithms, meaning processes do not have access to local randomness. We analyze the impact of our model on Byzantine consensus at different resilience thresholds: $n = 3f + 1$, $n = 2f + 1$, and $n = f + 2$, where n is the total number of processes, and up to f processes may be faulty.

We find that our model enables consensus protocols that are impossible under standard asynchrony. The key insight is that the random asynchronous model prevents an adversary from blocking honest parties from communicating indefinitely. In traditional asynchrony, an adversary can delay messages indefinitely to prevent termination (e.g., in the FLP impossibility result [13]). In contrast, in the random asynchronous model, the adversary cannot control the schedule, ensuring that honest parties can exchange messages within a bounded number of steps with high probability.

This last property is reminiscent of what synchrony guarantees, i.e., that any message sent by a correct process is received within a time bound Δ. This leads to the natural question: *How powerful is the random asynchronous model with respect to the standard models – asynchrony, partial synchrony, and synchrony?* We answer this question in Section 8. In brief, we show that the random asynchronous model sits between asynchrony and synchrony in terms of task solvability power, yet perhaps surprisingly, is *incomparable* to partial synchrony. To illustrate the latter separation, consider the following two tasks: (1) deterministic consensus when at least one process may fail by crashing, and (2) Byzantine consensus for $n < 3f + 1$. Task (1) is not solvable in the random asynchronous model, as we show in Section 7, but is solvable in partial synchrony [9]; conversely, task (2) is not solvable in partial synchrony [12] yet becomes attainable in the random asynchronous model with probabilistic safety (Section 5). Finally,

we also show that the random asynchronous model behaves like a variant of synchrony where timing guarantees hold with high probability instead of deterministically, thus confirming the intuitive similarity between the two models.

1.1 Modeling Challenges

Designing an asynchronous model with a non-adversarial schedule presented several challenges. Our initial attempt at a round-based model, while mathematically tractable, proved too rigid, enforcing communication patterns that were overly restrictive. We then explored probabilistic scheduling over entire message schedules, but this approach was unintuitive and impractical for analysis. A more promising approach was to randomly select individual messages for delivery. However, this exposed a critical vulnerability: Byzantine processes could manipulate the scheduling distribution by flooding the system with messages, effectively regaining control over the schedule. Attempts to mitigate this by encrypting messages failed, as the adversary could still infer crucial protocol information from the message patterns and delivery timings. More details about our initial attempts to model random asynchrony can be found in Section A.1.

These challenges led us to our final model: instead of selecting individual messages, the scheduler randomly selects sender-receiver pairs. At each step, a sender-receiver pair (s, r) is chosen, and the earliest pending message from s to r is delivered. This approach prevents Byzantine nodes from biasing the schedule since the probability of a message being delivered depends only on the number of available sender-receiver pairs, not on the volume of messages sent by a single process.

A natural concern is whether removing adversarial scheduling trivializes the consensus problem. We address this in two ways. First, Appendix A.2 presents a naive algorithm that fails to solve consensus when $n \leq 2f + 1$, highlighting a fundamental challenge: even though the random asynchronous model guarantees eventual communication, Byzantine nodes can still equivocate, preventing honest processes from distinguishing between correct and faulty messages. Second, we complement our positive results with corresponding impossibility results, establishing close bounds on what is solvable in the random asynchronous model.

1.2 Our Results

Our key contribution is the introduction of the random asynchronous model, a novel relaxation of the classic asynchronous model that removes adversarial scheduling while preserving unbounded message delays and Byzantine faults. This relaxation allows us to achieve new bounds that were previously impossible under standard asynchrony. We then design BFT consensus protocols in this new model, analyzing their feasibility at different resilience thresholds. Finally, we provide both positive results (demonstrating feasibility) and impossibility results (establishing the model's limits), summarized in Table 1.

2 Related Work

Random scheduling. Similar assumptions to the random asynchronous model have been explored by previous work. In message passing, Bracha and Toueg [8] define the fair scheduler, which ensures that in each message round, there is a non-zero constant probability that every correct process receives messages from the same set of correct processes. Under this scheduler, they proposed deterministic asynchronous binary consensus protocols for crash and Byzantine fault models. More recently, Tusk [11] and Mahi-Mahi [16] employ a form of

Table 1 Guarantees achievable (top) and impossible (bottom) in the random asynchronous model. "Strong" and "weak" validity are defined in Section 3.

n versus f	**Validity**	**Agreement**	**Termination**
	Positive results		
$n = 3f + 1$	Deterministic strong	Deterministic	Almost surely
$n = 2f + 1$	With high probability strong	With high probability	Deterministic
$n = f + 2$	Deterministic weak	With high probability	Deterministic
	Negative results		
$n = 3f + 1$	Deterministic strong	Deterministic	Deterministic
$n = 2f + 1$	Almost surely strong	Almost surely	Deterministic
$n = f + 2$	With high probability strong	With high probability	Deterministic

random scheduling. Their random scheduler can be seen as a special case of ours: they only consider the $n = 3f + 1$ case and assume a standard round-based model with the subset of processes that a process "hears from" in a given round chosen *uniformly* at random among all possibilities. They leverage the random scheduler to increase the probability to commit at each round, and thus to reduce latency, whereas our paper focuses on circumventing impossibility results in standard asynchrony, at different ratios of fault-tolerance. They conduct experiments without randomization on a wide-area network, without observing loss of liveness, which can serve as motivation for our work.

In shared memory, Aspnes [4] defines noisy scheduling, in which the adversary may choose the schedule, but that adversarial schedule is perturbed randomly. Under this assumption, deterministic asynchronous consensus becomes achievable. Also in shared memory, previous work introduce a *stochastic scheduler* [2, 3], which schedules shared memory steps randomly. This line of work shows that many lock-free algorithms are essentially wait-free when run against a stochastic scheduler, because the schedules that would break wait-freedom have negligible probability.

Randomized consensus. A large body of research leverages random coins to circumvent the FLP impossibility result [13], which states that deterministic consensus is impossible in crash-prone asynchronous systems. In this approach, protocols relax deterministic termination to probabilistic termination, assuming processes have access to a source of (cryptographically-secure) randomness that cannot be predicted by the adversary. Randomized consensus protocols employ either local coins [7, 21, 28] – which produce randomness independently and locally at each process, without coordination with other processes – or common coins [10, 11, 16, 22, 23, 25] – which, through the use of coordination and strong cryptographic primitives, ensure that all correct processes receive the same random output with some probability.

Our algorithms for the $n = 3f + 1$ setting resemble existing coin-based random consensus protocols, with the randomness moved from the process logic to the schedule. In fact, all of the coin-based consensus protocols we examined can be transformed, with minor changes, into deterministic (coin-less) algorithms in the random asynchronous model. A natural question, then, is whether our model is equivalent to the standard asynchronous model with coin tosses. It is not: in the standard model, achieving safety *w.h.p.* when $n < 3f$ is impossible, even if processes have access to randomness. This is because of a standard network partition argument: the adversary can partition correct processes into two sets that

never exchange messages, allowing Byzantine to force different decisions in each set. By contrast, our model makes long-lived network partitions occur with negligible probability, allowing safety *w.h.p.* even for $n < 3f$.

Probabilistic quorum systems. A line of work on probabilistic quorum systems [20, 27] relaxes quorum intersection to be probabilistic rather than deterministic, and allows for probabilistic correctness guarantees. However, they are vulnerable to an adversarial scheduler [1]. ProBFT [5] addresses this through the use of verifiable random functions for quorum selection. These works are similar to ours in that correctness is probabilistic rather than deterministic, but their approach focuses on the $n = 3f + 1$ setting, and is mainly aimed at scalability and efficiency (e.g., communication complexity), whereas we aim to circumvent impossibilities in standard asynchrony across various fault tolerance ratios.

3 Model & Preliminaries

Processes. We consider a set Π of n processes, up to f of which may be faulty. We consider the Byzantine-fault model, in which faulty processes may depart arbitrarily from the protocol. Throughout the paper, we assume that correct (non-faulty) processes have deterministic logic, i.e., they do not have access to a source of randomness.

Cryptography. We make assumptions that are standard for Byzantine fault-tolerant algorithms: processes communicate through authenticated channels (e.g., using message authentication codes (MACs)) and have access to digital signatures whose properties cannot be broken by the adversary.

Network. The processes communicate by sending messages over a fully-connected reliable network: every pair of processes p and q communicate over a link that satisfies the *integrity* and *no-loss* properties. Integrity requires that a message m from p be received by q at most once and only if m was previously sent by p to q. No-loss requires that a message m sent from p to q be eventually received by q.

Random asynchrony. Processes send messages by submitting them to the network; a *random scheduler* decides in which order submitted messages are delivered. We assume the scheduler delivers messages one by one, i.e., no two delivery events occur at exactly the same time. For convenience, we use a global discrete notion of time to reflect the sequence of delivery events: time proceeds in discrete steps $0, 1, \ldots$; each time step corresponds to a message delivery event in the system. Processes do not have access to this notion of time (they do not have clocks). Sending a message and performing local computation occur instantaneously, between time steps. Note that this notion of time does not bound message delays: an arbitrary amount of real time can pass between time steps.

We next describe the random scheduler. At any time t, let $P(t) \subseteq \Pi^2$ be the set of pairs of processes (p, q) such that p has at least one *pending message* to q. We say that a message m from p to q is pending at time t if p sends m before t and q has not received m by time t. At each time step t, the random scheduler draws a pair (p, q) at random from $P(t)$ and delivers to q the earliest message from p. We assume that there exists a lower bound $\mathcal{C}(n, f) > 0$ such that, for any p and q, the probability that $(p, q) \in P(t)$ is drawn at time t is at least $\mathcal{C}(n, f)$. In general, $\mathcal{C}(n, f)$ may depend on n and f, but not on t. We assume that scheduling draws do not depend on the content of the message being delivered.

More formally, let $\{X_t\}_{t\geq 0} \in P(t)$ be the random variables that represent the scheduling draws at each step t, and let $\mathcal{F}_t$ be the history (all messages sent/delivered) up to step t. Then our lower bound assumption above can be written as

$$\Pr[X_t = (p,q) \mid \mathcal{F}_t] \;\geq\; \mathcal{C}(n,f) \quad \text{for any} \quad t \geq 0 \tag{1}$$

Consensus. Our algorithms solve two variants of binary Byzantine consensus: strong and weak. Strong Byzantine consensus is defined by the following properties:
Strong Validity If all correct processes propose v, then correct processes that decide, decide v.
Agreement If correct processes p and q decide v and w respectively, then $v = w$.
Termination Every correct process decides some value.

Weak Byzantine consensus [18] has the same agreement and termination properties as the strong variant above, but has a different validity property:
Weak Validity If all processes are correct and propose v, then processes that decide, decide v.

Deterministic and probabilistic properties. The probability of a schedule is the probability of the intersection of all its scheduling steps. We say that an algorithm $\mathcal{A}$ ensures a property $\mathcal{P}$ *deterministically* if $\mathcal{P}$ holds in every execution of $\mathcal{A}$. Note that any algorithm that ensures a property $\mathcal{P}$ deterministically in the standard asynchronous model, must also ensure $\mathcal{P}$ deterministically in the random asynchronous model. Given an algorithm $\mathcal{A}$ and a property $\mathcal{P}$, we say that a schedule S is *bad for* $\mathcal{P}$ if there exists an execution E of $\mathcal{A}$ with schedule S such that $\mathcal{P}$ does not hold in E. An algorithm $\mathcal{A}$ ensures a property $\mathcal{P}$ *almost surely (a.s.)* if the total probability of all schedules that are bad for $\mathcal{P}$ is 0. An algorithm $\mathcal{A}$ ensures a property $\mathcal{P}$ *with high probability (w.h.p.)* if the total probability of all schedules that are bad for $\mathcal{P}$ is negligible; in this paper, a probability is negligible if it approaches 0 exponentially with some parameter of the algorithm, for any (fixed) n and f (e.g., the number of communication steps).

4 $n = 3f + 1$: Deterministic Safety, Termination Almost Surely

In this section we solve binary Byzantine consensus with deterministic strong validity and agreement, and termination almost surely.

Our algorithm, shown in Algorithm 1, proceeds in rounds: in each round, processes attempt to decide using a ROUND procedure; if unsuccessful, processes update their estimate and try again in the next round. The ROUND procedure is similar to an adopt-commit object [14, 26]: processes *propose* a value and return a pair (g, v), where v is a value and g is a grade, which can be either COMMIT or ADOPT. If $g =$ COMMIT, then it is guaranteed that all correct processes return the same value v (possibly with different grades). If all correct processes propose the same value v, then all correct processes are guaranteed to return v with a COMMIT grade.

The core of the algorithm is the ROUND procedure, shown in Algorithm 2. There are two types of messages: INIT and ECHO. Processes rely on Byzantine Reliable Broadcast (BRB) [9] for communication.[1] Furthermore, all messages are signed. The algorithm has two phases. In phase 1, each correct process p broadcasts (using BRB) its input value v in an INIT message (line 2), and waits to deliver $n - f = 2f + 1$ INIT messages (line 3). Process

[1] Any asynchronous BRB protocol is also correct in the random asynchronous model.

Algorithm 1 Main consensus algorithm for $n = 3f + 1$: pseudocode at process i.

```
1: procedure PROPOSE(v_i):                                  ▷ v_i ∈ {0, 1}
2:     est_i ← v_i
3:     r_i ← 0
4:     while true:
5:         (g, v) ← ROUND(r_i, est_i)
6:         if g = COMMIT: decide(v)                         ▷ Only once
7:         est_i ← v
8:         r_i ← r_i + 1
```

Algorithm 2 Byzantine ROUND implementation for $n = 3f + 1$: pseudocode at process i.

```
1: procedure ROUND(r, v):
2:     BRB-Broadcast ⟨INIT, r, v⟩                           ▷ Phase 1
3:     Wait to BRB-Deliver ⟨INIT, r, _⟩ from n − f processes
4:     H ← delivered ⟨INIT, r, _⟩ messages
5:     proposal ← majority value in H
6:     BRB-Broadcast ⟨ECHO, r, proposal, H⟩ to all processes  ▷ Phase 2
7:     Wait to BRB-Deliver valid ⟨ECHO, r, _, _⟩ messages from n − f processes
8:     if ∃ v* ≠ ⊥ such that I received ≥ 2f + 1 ⟨ECHO, r, v*, _⟩ messages:
9:         return ⟨COMMIT, v*⟩
10:    else:
11:        v* ← majority value among received ⟨ECHO, r, _, _⟩ messages
12:        return ⟨ADOPT, v*⟩
```

p adopts as its *proposal* for phase 2, the majority value among the received INIT messages (line 5). Then, in phase 2, p broadcasts an ECHO message with p's phase 2 proposal, as well as the $n - f$ (signed) INIT messages that justify v to be the majority phase 1 value (line 6); p waits for $n - f$ valid ECHO messages (line 7). If all delivered ECHO messages are for the same value v^*, then p commits v^* (line 9); otherwise p adopts the majority value (line 12).

We prove the correctness of our algorithm in Section B. The main intuition is that at each round, a favorable schedule can help all correct processes agree (i.e., adopt the same estimate), by causing them to select the same majority value in phase 1 (e.g., by ensuring that they deliver INIT messages from the same set of processes). Since the schedule is random, it has a non-zero chance of being favorable at each round. Thus, almost surely, the schedule will eventually be favorable. This is similar in spirit to random coin based Byzantine Randomized Consensus algorithms [9], in which correct processes choose their next estimate by tossing a coin, if they do not manage to reach agreement in a given round. Here we are relying on the random schedule instead of the random coin. However, as we show later, in our model, consensus is solvable for settings where it is impossible for standard asynchrony even when equipped with a common coin.

Crash-fault tolerant consensus. A similar approach can be used to solve crash-fault tolerant consensus with $n = 2f + 1$, with the same guarantees of deterministic safety, as well as termination almost surely. Our algorithm uses the same round-based structure in Algorithm 1 and a modified ROUND procedure, that does not require reliable broadcast, and employs standard point-to-point messages instead. The main intuition is similar to our Byzantine algorithm: the random scheduler eliminates the need for a common coin by ensuring that correct processes eventually deliver messages in a favorable order which leads them to agree and thus terminate. Section C gives our algorithm and proves its correctness.

5 $n = 2f + 1$: Deterministic Termination, Safety with High Probability

In this section, we pose $n = 2f + 1$ and solve strong Byzantine consensus such that safety (validity and agreement) holds with high probability and termination is deterministic.

Algorithm 3 shows our proposed protocol. The main idea is as follows: There are $f + 1$ phases, each consisting of R communication rounds, where R is a parameter. R is large enough so that correct processes hear from each other at least once within R rounds with high probability. In each round, a correct process sends its set of *accepted values* (V_i in Algorithm 3) to all other processes. A correct process i *accepts* a value from process j at phase p ($p = 1, \ldots, f + 1$) if (1) j is the origin of v (i.e., the first signature on v is by j), (2) i has not already accepted a value from j, and (3) v has valid signatures from p distinct processes. We say that i accepts a value v at phase p if p is the earliest phase at which i accepts v (from any process).

After the $f + 1$ phases are over, a correct process decides on the majority value within its set of accepted values (i.e., the value that appears most often). Note that at the end of the communication phases, each correct process must have at least one accepted value, since correct processes sign and send their input value in phase 1 and the network is reliable.

The main intuition behind this protocol is that, if a correct process accepts a value v, then all correct processes will accept v by the end of the execution, and thus all correct processes will have the same set of accepted values. Therefore, all correct processes can use a deterministic rule to decide the same value.

Algorithm 3 Binary Byzantine consensus for $n = 2f + 1$; pseudocode for process i.

```
1:  Local variables:
2:    V_i = ∅, a map from processes to signed values

3:  procedure PROPOSE(v):
4:    V_i[i] ← SIGN(v)
5:    for phase in 1 ... f + 1:
6:      for round in 1 ... R:
7:        Send (V_i, phase, round) to all processes
8:        valid ← 0
9:        while valid < n − f:
10:         Receive a (V_j, p, r) message ▷ receive single msg per process, phase and round
11:         if ∃v ∈ V_j : v is signed by phase distinct processes and V_i[ORIGIN(v)] = ∅
12:           V_i[ORIGIN(v)] ← SIGN(v)
13:         if (p, r) = (phase, round)
14:           valid ← valid + 1
15:   decide MAJORITYVALUE(V_i)
```

We now prove that Algorithm 3 satisfies the consensus properties in Section 3.

▶ **Lemma 1.** *With high probability, every correct process receives at least one message from every other correct process in each phase.*

Proof. We start by fixing two correct processes p and q and upper bounding the probability that q does not receive any message from p after R iterations. The probability of not delivering p's round-1 message to q is at most $(1 - \mathcal{C}(n, f))$ at each time step. There must be

at least $R(n-f)$ time steps for q to complete R iterations, since q waits for at least $n-f$ messages at each iteration, and each message received consumes one time step. Thus, using assumption (1), the probability of q not observing p's message is at most

$$(1-\mathcal{C}(n,f))^{R(n-f)} \approx e^{-R\mathcal{C}(n,f)(n-f)}.$$

To finish the proof, we upper-bound the probability of any correct process not observing the input value of some other correct process. We first compute the expected number of (ordered) pairs of processes (p,q) such that q does not observe the input value of p at the end of the R iterations. There are $n(n-1)$ possible pairs, so this expected value is at most $E = n(n-1)e^{-R\mathcal{C}(n,f)(n-f)}$. Now, by Markov's inequality, we have that

$$\Pr(\text{at least one unreachable pair}) \leq E = n(n-1)e^{-R\mathcal{C}(n,f)(n-f)}.$$

For fixed n and f, this probability approaches 0 exponentially in R. ◀

▶ **Lemma 2.** *With high probability, every correct process accepts the input values of every other correct process.*

Proof. By Lemma 1, every correct process receives at least one message from every other correct process in the first phase, w.h.p. Since messages from correct processes always contain their input values with a valid signature, every correct processes i will accept the input value of another correct process j when i receives the first message from j. ◀

▶ **Theorem 3.** *With $n = 2f+1$, Algorithm 3 satisfies strong validity w.h.p.*

Proof. Assume that all correct processes propose the same value v. Then, by Lemma 2, all correct processes will accept at least $n-f = f+1$ copies of value v w.h.p. Since $f+1$ is a majority out of a maximum of $n = 2f+1$ accepted values, correct processes decide v w.h.p. ◀

▶ **Lemma 4.** *If a value v is accepted by a correct process i at phase $p \leq f$, then all correct processes accept v by the end of phase $p+1$, w.h.p.*

Proof. If i accepts v at phase p, then v must have signatures from at least p processes. Process i is not one of the p processes, otherwise i would have accepted v at an earlier phase. Since i accepts v, i adds its signature to v and will send v, as part of V_i, to all processes in phase $p+1$. By Lemma 1, all correct processes will thus receive v by the end of phase $p+1$ w.h.p., and will accept v (if they haven't already), since v has the required number of signatures. ◀

▶ **Lemma 5.** *If a value v is accepted by a correct process i at phase $f+1$, then all correct processes accept v by the end of phase $f+1$, w.h.p.*

Proof. If i accepts v at phase $f+1$, then v must have signatures from at least $f+1$ processes; i is not among these processes, otherwise i would have accepted v at an earlier phase. Since there are at most f faulty processes, v must have at least one signature from a correct process $j \neq i$.

So j must have accepted v at an earlier phase $p \leq f$ and therefore, by Lemma 4, all correct processes will accept v by the end of phase $f+1$ w.h.p. ◀

▶ **Theorem 6.** *With $n = 2f+1$, Algorithm 3 satisfies agreement w.h.p.*

Proof. By Lemma 4 and Lemma 5, with high probability, correct processes have the same set of accepted values by the end of phase $f + 1$, and thus decide the same value. ◀

▶ **Theorem 7.** *With $n = 2f + 1$, Algorithm 3 satisfies deterministic termination.*

Proof. Follows immediately from the algorithm: correct processes only execute for $R(f + 1)$ rounds. In each round, a correct process waits to receive $n - f$ messages from that round, which is guaranteed to occur in a finite number of steps, since there are at least $n - f$ correct processes and the network is reliable (no-loss property). ◀

6 $n = f + 2$: Deterministic Termination and Validity, Agreement w.h.p.

Interestingly, for $n = f + 2$, we can solve weak Byzantine consensus with deterministic validity and termination, and agreement *w.h.p.*, using the same protocol in Algorithm 3.

Weak validity is clearly preserved: if all processes are correct and have the same input value v, no other value is received by any process, and thus all processes decide v.

▶ **Theorem 8.** *With $n = f + 2$, Algorithm 3 satisfies deterministic weak validity.*

Proof. If all processes are correct and propose the same value v, then all $(V_i, phase, round)$ messages will have v as their value, so no process can decide any other value. ◀

▶ **Theorem 9.** *With $n = f + 2$, Algorithm 3 satisfies agreement w.h.p.*

Proof. Lemmas 1, 2, 4, and 5 still hold: their proofs are also valid if $n = f + 2$. Thus the proof of this theorem is the same as the proof of Theorem 6: By Lemma 4 and Lemma 5, with high probability, correct processes have the same set of accepted values by the end of phase $f + 1$, and thus decide the same value. ◀

▶ **Theorem 10.** *With $n = f + 2$, Algorithm 3 satisfies deterministic termination.*

Proof. Correct processes only execute for $R(f + 1)$ rounds. In each phase and round, a correct process waits to receive $n - f$ valid messages from that phase and round. This wait is guaranteed to terminate since there are at least $n - f$ correct processes, correct processes can always produce a valid message (a message (V_i, p, r) is valid if p and r are equal to the current phase and round, respectively), and the network is reliable (no-loss property). ◀

7 Negative Results

In this section we provide negative results that closely match our positive results from previous sections. Intuitively, we show that in the random asynchronous model it is not possible to obtain Byzantine consensus protocols with more powerful guarantees than the protocols we propose in this paper. This shows that the random asynchronous model, while avoiding some restrictions and impossibilities of the standard asynchronous model, is not overly permissive. Concretely, we prove the following three results.

▶ **Theorem 11.** *No protocol can solve consensus in the random asynchronous model with deterministic strong validity, agreement, and termination, if at least one process may fail by crashing.*

▶ **Theorem 12.** *With $n = 2f + 1$, no protocol for Byzantine consensus in the random asynchronous model can ensure strong validity and agreement with probability 1, as well as deterministic termination.*

▶ **Theorem 13.** *With $n = f + 2, f \geq 2$, no protocol for Byzantine consensus in the random asynchronous model can ensure strong validity and agreement with high probability, while ensuring deterministic termination.*

Proof sketch for Theorem 11. This result is equivalent to the FLP impossibility [13] in the random asynchronous model, and the FLP proof holds in our model as well. Essentially, if at least one process can fail by crashing, there exists an infinite bivalent execution (the same execution as constructed in the FLP proof), which prevents processes from deciding without breaking agreement. ◀

Instead of proving Theorem 12 directly, we can prove the following stronger result, which also implies Theorem 12.

▶ **Theorem 14.** *No protocol for consensus in the random asynchronous model can ensure strong validity and agreement almost surely, as well as deterministic termination, if at least one process may fail by crashing.*

Proof. Assume such a protocol $\mathcal{A}$ exists. Since $\mathcal{A}$ ensures deterministic termination, it must terminate in finite time in every execution. Furthermore, $\mathcal{A}$ ensures strong validity and agreement almost surely, so the probability of schedules in which $\mathcal{A}$ terminates in a bivalent state (in the sense of FLP [13]) must be 0. Yet, following the FLP proof, there exists an infinite bivalent execution E. It follows that $\mathcal{A}$ must decide in a finite prefix π of E, when the state is still bivalent. Since π has finite length, its schedule has non-zero probability. We have shown that $\mathcal{A}$ does not ensure agreement and validity almost surely, a contradiction. ◀

Proof of Theorem 13. Assume towards a contradiction that such a protocol $\mathcal{A}$ exists. Call an execution E of $\mathcal{A}$ *heterogeneous* if in E at least 2 processes propose 0 and at least 2 processes propose 1. Assume *wlog* that p_1 and p_2 propose 0, while p_{n-1} and p_n propose 1.

Let E be a heterogeneous execution in which all processes are correct. Let S be the schedule of E. Let π be shortest prefix of S in which all processes have decided; in other words, we truncate any steps in S after the processes have decided. The prefix π must be finite since $\mathcal{A}$ ensures deterministic termination.

We now describe three executions E_1, E_2, and E_3, with the same schedule π, and show that $\mathcal{A}$ must break either strong validity or agreement in one of the executions. This is sufficient to show that π is bad (in the sense of Section 3) for strong validity or agreement.

Let E_1 be an execution with schedule π, in which all processes behave identically to E; in E_1, p_1 and p_2 are correct, while $p_3, \ldots, p_n$ are Byzantine but behave correctly. By strong validity, p_1 must decide 0 in E_1.

Let E_2 be an execution with schedule π, in which again all processes behave identically to E; this time, p_1 and p_n are correct, while $p_2, \ldots, p_{n-1}$ are Byzantine. Since E_1 and E_2 are indistinguishable to p_1, and p_1 has deterministic logic, p_1 must decide the same value in both executions, namely 0. Thus, in order to satisfy agreement, p_n must also decide 0 in E_2.

Let E_3 be an execution with schedule π, in which again all processes behave as in E; this time, p_{n-1} and p_n are correct, while $p_1, \ldots, p_{n-2}$ are Byzantine. Since E_2 and E_3 are indistinguishable to p_n, and p_n has deterministic logic, p_n must decide the same value in both executions, namely 0. But this breaks strong validity, as both correct processes (p_{n-1} and p_n) have proposed 1 in E_3.

We have shown that π is bad for agreement or strong validity. Since the length of π is fixed (and finite) for fixed n and f, the probability of π is non-negligible according to our definition in Section 3. Therefore, $\mathcal{A}$ does not ensure strong validity and agreement with high probability, a contradiction. ◀

8 The Random Asynchronous Model vs Standard Models

In this section we compare the random asynchronous model with the standard network models – asynchrony, synchrony, and partial synchrony – in terms of task solvability. We show that it is **(i) strictly stronger** than full asynchrony (i.e., the set of tasks solvable in random asynchrony is a strict superset of the set of tasks solvable in asynchrony), **(ii) strictly weaker** than synchrony, and **(iii) incomparable** with partial synchrony, with respect to task solvability. Table 2 summarizes the relationships.

Table 2 Relative power of the random asynchronous model.

	Asynchrony	Partial synchrony	Synchrony
Random asynchrony	strictly stronger	incomparable	strictly weaker

Asynchrony. Every task solvable in asynchrony is also solvable in random asynchrony because *every finite prefix of an asynchronous schedule occurs with non-zero probability* under our scheduler.[2] Conversely, the task "Byzantine binary consensus with $n = 2f + 1$, deterministic termination, and safety holding with high probability" is solvable in the random asynchronous model (Section 5) yet impossible in pure asynchrony by the standard network partition argument [7, 8]. Hence the random asynchronous model is strictly stronger.

Synchrony. The random asynchronous model is strictly more restrictive than synchrony: any task that is solvable in the random asynchronous model is also solvable in synchrony, and there are tasks which are solvable in synchrony but not in the random asynchronous model. As an example of the latter, consider deterministic crash-fault tolerant consensus: this task is impossible in the random asynchronous model (Theorem 11) but it is solvable in synchrony.

We now prove the former direction: any task solvable in the random asynchronous model is also solvable in synchrony. Take any task τ and let $\mathcal{A}_{RA}$ be a protocol that solves τ in the random asynchronous model. We can simulate $\mathcal{A}_{RA}$ in synchrony using an adapter Σ_p for each process p between $\mathcal{A}_{RA}$ and the network, in the following way:

1. At each synchronous time step $t = 0, 1, \ldots$, each correct process p executes all outstanding local steps of $\mathcal{A}_{RA}$, including those triggered by messages delivered since the previous time step. Process p sends the same messages it would in $\mathcal{A}_{RA}$, by handing them to Σ_p.
2. The synchronous network delivers all messages send at time t by time $t + 1$. Σ_p buffers any messages received by p between t and $t + 1$.
3. At time $t + 1$, Σ_p draws a fresh random permutation of the links on which p has received messages, then delivers to $\mathcal{A}_{RA}$ the *earliest* pending message on each link in that order.

[2] Formally, let S be any (possibly infinite) asynchronous schedule and π a finite prefix of S. Since each delivery step is chosen independently with lower-bound probability $C(n, f) > 0$, the probability of π is at least $C(n, f)^{|\pi|}$.

The resulting schedule visible to $\mathcal{A}_{RA}$ is valid in the random asynchronous model, as any causal relation is preserved. Since the Σ_p adapters collectively deliver messages from at most n^2 links at each synchronous time step, each link has a probability of at least $1/n^2 := \mathcal{C}(n, f)$ of being selected at each time step internal to the random asynchronous protocol $\mathcal{A}_{RA}$. Thus, the Σ_p adapters preserve the probabilistic delivery properties of the random asynchronous model. We have described a construction that simulates any random asynchronous algorithm in synchrony, thus showing that any task that is solvable in the random asynchronous model is solvable in synchrony.

Partial synchrony. The random asynchronous model is incomparable with respect to partial synchrony in terms of solvability: there are tasks that are solvable in the random asynchronous model, but not in partial synchrony, and vice-versa.

As an example of a task that is solvable in the random asynchronous model but not in partial synchrony, recall the same $n = 2f + 1$ Byzantine consensus task above. Section 5 shows that this task is solvable in the random asynchronous model. However, this task is not solvable in partial synchrony [12].

To show that some tasks are solvable in partial synchrony but not in the random asynchronous model, consider the task of (deterministic) crash-fault tolerant consensus. This task is solvable in partial synchrony if $n \geq 2f + 1$ [9], but is not solvable in the random asynchronous model if at least one process can crash (Theorem 11).

Discussion. Although the random–asynchronous model is *strictly* weaker than full synchrony in terms of task solvability, it can approximate synchronous behavior with high probability. After R rounds of all-to-all communication, every pair of correct processes has exchanged at least one message with probability $1-\exp(-\Omega(R))$ (Lemma 1). This mirrors the deterministic guarantee offered by a single synchronous round – modulo the known time bound Δ.

Leveraging this property, we can implement, in the random asynchronous model, a crash-fault detector $\sim\mathcal{P}$ that is *perfect with high probability*:

Strong completeness Every process that crashes is eventually permanently suspected by every correct process.

Strong accuracy w.h.p. No correct process is suspected by any correct process w.h.p.

The implementation is a classic heartbeat scheme. Each process broadcasts a heartbeat message once per round; process p suspects q iff it has not received a heartbeat from q in the last R rounds. As we increase R, the probability of false positive suspicion decreases exponentially, through an argument similar to Lemma 1.

Such a failure detector is similar to the perfect $\mathcal{P}$ and eventually perfect $\diamond\mathcal{P}$ failure detectors, which can be implemented in synchrony and partial synchrony respectively [9]. Whether protocols that rely on $\mathcal{P}$ or $\Diamond\mathcal{P}$ can be systematically translated to use $\sim\mathcal{P}$ – preserving their guarantees w.h.p.– remains an open question.

Intuitively, such a systematic translation seems unlikely to exist, as downgrading deterministic communication guarantees to hold w.h.p. could break arbitrary internal algorithm invariants. Consider a synchronous algorithm that runs for T rounds; such an algorithm is able to rely on the fact that it will receive messages from the same set of at least $n - f$ correct processes in each of the T rounds – e.g., at the end of round T each process p may output the $(n - f)^{\text{th}}$ highest process id p heard from in every one of the T rounds. However, if we directly translate this algorithm to random asynchrony, by replacing each synchronous round with R asynchronous rounds (for at total of RT rounds), the guarantee of receiving messages from a common subset of $n - f$ processes holds merely with high

probability, not deterministically. In the rare executions where p hears from a common subset of fewer than $n - f$ processes, the translated algorithm would try to access data that does not exist, violating its invariants. Any translation from synchrony to random asynchrony would therefore need manual, algorithm-specific fallback logic to cope with those low-probability outliers, making a generic translation unlikely.

9 Conclusion

We introduce the random asynchronous model, a novel relaxation of the classic asynchronous model that replaces adversarial message scheduling with a randomized scheduler. By eliminating the adversary's ability to indefinitely delay messages, our model circumvents traditional impossibility results in asynchronous Byzantine consensus while preserving unbounded message delays and tolerating Byzantine faults. Our approach avoids the need for synchronized periods (as in partial synchrony) or cryptographic randomness (as in randomized consensus), offering a foundation for practical alternatives to existing asynchronous systems. We demonstrated that this relaxation enables new feasibility results across different resilience thresholds: deterministic safety and probabilistic termination for $n = 3f + 1$, deterministic termination with safety holding with high probability (w.h.p.) for $n = 2f + 1$, and weak validity with w.h.p. agreement for $n = f + 2$. These results are complemented by impossibility bounds, showing our protocols achieve near-optimal guarantees under the model.

Future work could explore extensions of this model to other distributed computing problems, such as state machine replication, and investigate empirical performance trade-offs in real-world deployments. By bridging the gap between theoretical impossibility and practical assumptions, we believe our model opens avenues for more efficient and resilient distributed protocols.

References

1 Amitanand S. Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. On the availability of non-strict quorum systems. In *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724, pages 48–62. Springer, 2005. doi:10.1007/11561927_6.

2 Dan Alistarh, Keren Censor-Hillel, and Nir Shavit. Are lock-free concurrent algorithms practically wait-free? *J. ACM*, 63(4):31:1–31:20, 2016. doi:10.1145/2903136.

3 Dan Alistarh, Thomas Sauerwald, and Milan Vojnovic. Lock-free algorithms under stochastic schedulers. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 251–260. ACM, 2015. doi:10.1145/2767386.2767430.

4 James Aspnes. Fast deterministic consensus in a noisy environment. *J. Algorithms*, 45(1):16–39, 2002. doi:10.1016/S0196-6774(02)00220-1.

5 Diogo Avelas, Hasan Heydari, Eduardo Alchieri, Tobias Distler, and Alysson Bessani. Probabilistic byzantine fault tolerance. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 170–181. ACM, 2024. doi:10.1145/3662158.3662810.

6 Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. In *AFT*, 2019.

7 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983. doi:10.1145/800221.806707.

8 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985. doi:10.1145/4221.214134.

9 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

10 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, pages 123–132. ACM, 2000. doi:10.1145/343477.343531.

11 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

12 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

13 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983*, pages 1–7. ACM, 1983. doi:10.1145/588058.588060.

14 Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, June 28 - July 2, 1998*, pages 143–152. ACM, 1998. doi:10.1145/277697.277724.

15 Giacomo Giuliari, Alberto Sonnino, Marc Frei, Fabio Streun, Lefteris Kokoris-Kogias, and Adrian Perrig. An Empirical Study of Consensus Protocols' DoS Resilience. In *ACM ASIACCS*, 2024.

16 Philipp Jovanovic, Lefteris Kokoris Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. Mahi-mahi: Low-latency asynchronous bft dag-based consensus. *arXiv preprint arXiv:2410.08670*, 2024. doi:10.48550/arXiv.2410.08670.

17 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All You Need is DAG. In *ACM PODC*, 2021.

18 Leslie Lamport. The weak byzantine generals problem. *J. ACM*, 30(3):668–676, 1983. doi:10.1145/2402.322398.

19 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

20 Dahlia Malkhi, Michael K. Reiter, Avishai Wool, and Rebecca N. Wright. Probabilistic quorum systems. *Inf. Comput.*, 170(2):184–206, 2001. doi:10.1006/INCO.2001.3054.

21 Henrique Moniz, Nuno Ferreira Neves, Miguel Correia, and Paulo Veríssimo. RITAS: services for randomized intrusion tolerance. *IEEE Trans. Dependable Secur. Comput.*, 8(1):122–136, 2011. doi:10.1109/TDSC.2008.76.

22 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $o(n^2)$ messages, and $o(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, 2015. doi:10.1145/2785953.

23 Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.48.

24 Zhijie Ren, Kelong Cong, Johan Pouwelse, and Zekeriya Erkin. Implicit Consensus: Blockchain with Unbounded Throughput, 2017. arXiv:1705.11046.

25 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

26 Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 297–306. ACM, 1998. doi:10.1145/277697.277755.

27 Haifeng Yu. Signed quorum systems. *Distributed Comput.*, 18(4):307–323, 2006. doi:10.1007/S00446-005-0133-8.

28 Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. Waterbear: Practical asynchronous BFT matching security guarantees of partially synchronous BFT. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 5341–5357. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-haibin.

APPENDIX

A Challenges

A.1 Modeling Challenge

Our aim is to propose a model for asynchrony without adversarial scheduling that is (1) general, i.e., does not restrict algorithm design (2) easy to work with for proofs, (3) usable by practical algorithms, and (4) intuitive. In the course of defining the current model, we came up with several other possibilities that do not meet the aims above:

1. A round-based model, similar to the fair scheduler model of Bracha and Toueg [8]. In each communication round, a correct process sends a message to every process and waits to hear back from $n - f$ processes. The random scheduler assumption is: in each round, a correct process has a non-trivial (i.e., lower-bounded by a constant) probability of hearing from any subset of $n - f$ processes. This model has the advantage of being easy to work with, but is too restrictive, as it restricts algorithms to the round-based structure.
2. A model which places probability directly on entire schedules, instead of on individual communication steps: each valid schedule has a non-trivial probability of occurring. We found this model to be un-intuitive and difficult to work with.
3. A model in which the next message to be delivered is drawn, according to some distribution, from all currently pending messages (i.e., messages that have been sent but not yet delivered). The distribution must ensure that every message has a non-trivial probability to be scheduled next. This model is general (does not restrict algorithm structure), intuitive, and easy to work with, but has the following crucial flaw. Byzatine processes can skew the scheduling distribution by producing a large number of messages (potentially under the guise of retransmitting them as part of the reliable links assumption). If, at any given time, most pending messages are from Byzantine processes, then these messages are more likely to be delivered first, effectively reverting the model to an adversarial scheduler.
4. Similarly to the previous proposal: at each scheduling step, a sender-receiver pair (s, r) is drawn *uniformly at random*, and the earliest pending message from s to r is the next message delivered in the system. This model is intuitive, and easy to work with, while also fixing the message injection attack by Byzantine processes: the number of pending messages from a (potentially Byzantine) process s to process r does not influence the probability distribution of s's messages to be delivered before other messages. The only drawback is with respect to generality: the uniform distribution on the sender-receiver pairs is a strong assumption.

Our final model is similar to the last proposal above, while solving the generality problem by removing the uniform distribution assumption. Instead, we simply assume that the probability of each sender-receiver pair being drawn is non-negligible.

A.2 Algorithmic Challenge

Take the following naive (and incorrect) binary consensus algorithm for the $n \leq 2f + 1$ cases ($n = 2f + 1$ and/or $n = f + 2$):

- Round 0: Processes initially sign and send their input value to all other processes, and wait for such messages from $n - f$ processes.
- Rounds 1–R (where R is a parameter): Processes sign and send their entire history of sent and received messages to all processes and wait for valid such messages from $n - f$ processes.
- At the end of round R, correct processes decide on, say, the lowest input value they have received.

This algorithm is subject to the following attack:

- Assume all correct processes have 1 as their input value.
- The f Byzantine processes do not send any messages to the $n - f$ correct processes up until and including round $R - 1$. Otherwise, Byzantine processes act as correct processes whose input values are 0, including accepting messages from correct processes.
- Thus, no correct process is aware of 0 as a valid input value before round R.
- Let p be some correct processes that the Byzantine processes have agreed on before the start of the execution. The attack attempts to cause p to decide on a different value than the other correct processes, breaking agreement.
- In round R, Byzantine processes send correctly constructed messages to p, containing their entire communication histories. If the random scheduler delivers even one of these messages to p before the end of the round, then p becomes aware of the input value 0 for the first time in round R (and therefore does not have time to inform the other correct processes).
- After round R, p must decide 0 (being the lowest input value it is aware of), while the other processes must decide 1 (not being aware of 0 as a valid input value), breaking agreement.

This attack succeeds if the random scheduler delivers a message from a Byzantine process to p in round R, among the first $n - f$ messages delivered to p in that round. This has a non-trivial chance of occurring.

This algorithm illustrates the main challenge of designing correct algorithms under the random asynchronous model when $n \leq 2f + 1$: even if the model ensures that all correct processes communicate with each other eventually, Byzantine processes can still equivocate and correct processes do not necessarily know which messages are from correct processes and which are not.

B Proofs for Deterministic Safety Algorithms

In this section we prove the correctness of our algorithm in Section 4. We first prove that the Round procedure in Algorithm 2 satisfies the properties below, and then prove that Algorithm 1 solves consensus under Byzantine faults.

Strong Validity If all correct processes propose the same value v and a correct process returns a pair $\langle \text{Grade}, v' \rangle$, then Grade = Commit and $v' = v$.

Consistency If any correct process returns $\langle \text{COMMIT}, v \rangle$, then no correct process returns $(\cdot, v' \neq v)$.

Termination If all correct processes propose, then every correct process eventually returns.

In our proofs we rely on the following properties of Byzantine Reliable Broadcast (BRB) [9]:

BRB-Validity If a correct process p broadcasts a message m, then every correct process eventually delivers m.

BRB-No-duplication Every correct process delivers at most one message.

BRB-Integrity If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BRB-Consistency If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.

BRB-Totality If some message is delivered by any correct process, every correct process eventually delivers a message.

▶ **Lemma 15.** *With Byzantine faults and $n = 3f + 1$, Algorithm 2 satisfies strong validity.*

Proof. If all correct processes propose the same value v, then at least $2f + 1$ processes BRB-broadcast an INIT message for v, and therefore at most f processes BRB-broadcast an INIT message for $1 - v$. Thus v will be the majority value among all INIT messages delivered in phase 1, at all correct processes. Thus all correct processes will BRB-broadcast an ECHO message for v. Furthermore, no Byzantine process can produce a valid ECHO message for $1 - v$, since to do so would require a set of $2f + 1$ INIT message with a majority value of $1 - v$. This is impossible due to the properties of BRB and the fact that at most f processes have BRB-broadcast an INIT message for $1 - v$. So, all valid ECHO messages received by correct processes will be for v, so all correct processes will commit v at line 9. ◀

▶ **Lemma 16.** *With Byzantine faults and $n = 3f + 1$, Algorithm 2 satisfies consistency.*

Proof. If a correct process p_1 commits v at line 9, then it must have delivered a set S_1 of $2f + 1$ ECHO messages for v at line 7. Take now another process p_2 and consider the set S_2 of $2f + 1$ ECHO messages it delivers at line 7. By quorum intersection, S_1 and S_2 must intersect in at least $f + 1$ messages. By the BRB-Consistency property, these $f + 1$ messages must be identical at p_1 and p_2. Thus p_2 delivers at least $f + 1$ ECHO messages for v, which constitutes a majority of the $2f + 1$ ECHO messages it delivers overall. So if p_2 commits a value at line 9, then it must commit v, and if p_2 adopts a value at line 12, then it must adopt v. ◀

▶ **Lemma 17.** *With Byzantine faults and $n = 3f + 1$, Algorithm 2 satisfies termination.*

Proof. Follows immediately from the algorithm and from the properties of Byzantine Reliable Broadcast. Processes perform two phases; the only blocking step of each phase is waiting for $n - f$ messages (lines 3 and 7). This waiting eventually terminates, by the BRB-Validity property and the fact that there are at least $n - f$ correct processes. ◀

▶ **Theorem 18.** *With Byzantine faults and $n = 3f + 1$, Algorithm 1 satisfies strong validity.*

Proof. This follows from the strong validity property of the ROUND procedure (Lemma 15): if all correct processes propose v to consensus, then all correct processes propose v to ROUND in the first round, where by Lemma 15, all correct processes commit v, and thus all correct processes decide v at line 6. ◀

▶ **Theorem 19.** *With Byzantine faults and $n = 3f + 1$, Algorithm 1 satisfies agreement.*

Proof. Let r be the earliest round at which some process decides and let p be a process that decides v at round r. We will show that any other process p' that decides, must decide v.

For p to decide v at round r, ROUND must output (COMMIT, v) in that round. Thus, by the consistency property of ROUND, $\text{ROUND}(r, \cdot)$ must output $(\cdot, v)$ at all correct processes. If $\text{ROUND}(r, \cdot)$ outputs (COMMIT, v) for p', then p' decides v at round r (line 6). Otherwise, all correct processes input v to $\text{ROUND}(r+1, \cdot)$, and by the strong validity property, all processes (including p') will output (COMMIT, v) and decide v at round $r+1$. ◀

▶ **Theorem 20.** *With Byzantine faults and $n = 3f + 1$, Algorithm 1 satisfies termination.*

Proof. We can describe the execution of the protocol as a Markov chain with states $0, \ldots, n - f = 2f + 1$; the system is at state i if i correct processes have estimate (est_i variable) equal to 0 before invoking ROUND. Due to the strong validity property of the ROUND procedure, states 0 and $2f + 1$ are absorbing states. There is a non-zero transition probability from each state (including 0 and $2f + 1$), to state 0 or $2f + 1$, or both (we show this below). Therefore, almost surely, the system will eventually reach one of the two absorbing states and remain there. Once this happens (i.e., once all processes have the same est_i variable), the strong validity property of ROUND ensures that all processes (who have not decided yet) will decide within a round.

It only remains to show that there is a non-zero transition probability from each state to at least one of the absorbing states 0 and $2f + 1$. Consider a state $i \notin \{0, 2f + 1\}$; there is a schedule S with non-zero probability which leads the system from i to 0 or $2f + 1$ in one invocation of ROUND. We consider two cases:

- $i < f + 1$: in this case 0 is the minority value among correct processes. In schedule S, the $n - f$ INIT messages delivered by correct processes at line 3 are all from correct processes. Thus, every correct process sees i 0s and $2f + 1 - i$ 1s; 1 is the majority value, so all correct processes adopt it for phase 2. In phase 2, S again ensures that the $n - f$ ECHO messages delivered by correct processes at line 7 are all from correct processes. Thus, all correct processes see $2f + 1$ ECHO messages for 1 and commit 1, bringing the system to state 0.
- $i \geq f + 1$: in this case 0 is the majority value among correct processes. This case is symmetrical with respect to the previous one: the only difference is that all correct processes adopt 0 (the majority value) at the end of phase 1, and all correct processes deliver $2f + 1$ ECHO messages for 0, thus committing 0 and bringing the system to state $2f + 1$. ◀

C Crash-fault Tolerant Consensus in the Random Asynchronous Model

C.1 Definition

In the crash-fault model, faulty processes may permanently stop participating in the protocol at any time, but otherwise follow the protocol. Crash-fault tolerant consensus is defined by the following properties:

Validity If a process decides v, then v was proposed by some process.

Uniform Agreement If processes p and q decide v and w respectively, then $v = w$.

Termination Every correct process decides some value.

C.2 Algorithm

Our algorithm for crash-fault tolerant consensus uses the same round-based structure, shown in Algorithm 1, as our Byzantine consensus algorithm from Section 4. We use a different implementation of the ROUND procedure, shown in Algorithm 4.

Algorithm 4 Crash-tolerant ROUND implementation: pseudocode at process i.

```
1:  procedure ROUND(r, v):
2:      Send ⟨INIT, r, v⟩ to all processes                          ▷ Phase 1
3:      Wait for n − f ⟨INIT, r, _⟩ messages
4:      if ∃ v* such that I received ≥ f + 1 ⟨INIT, r, v*⟩ messages:
5:          proposal ← v*
6:      else:
7:          proposal ← ⊥
8:      Send ⟨ECHO, r, proposal⟩ to all processes                   ▷ Phase 2
9:      Wait for n − f ⟨ECHO, r, _⟩ messages
10:     if ∃ v* ≠ ⊥ such that I received ≥ f + 1 ⟨ECHO, r, v*⟩ messages:
11:         return ⟨COMMIT, v*⟩
12:     else if ∃ v* ≠ ⊥ such that I received ≥ 1 ⟨ECHO, r, v*⟩ messages:
13:         return ⟨ADOPT, v*⟩
14:     else:
15:         v* ← value in first ⟨INIT, r, _⟩ message received
16:         return ⟨ADOPT, v*⟩
```

The ROUND algorithm consists of two phases. In the first phase, every correct process proposes a value by sending it to all processes (line 2). Then it waits to receive proposals from a quorum of processes (line 3). If a process observes that all responses contain the same phase-one proposal value then it proposes that value for the second phase (line 5). If a process does not obtain a unanimous set of proposals in the first phase, the process simply proposes $\perp$ for the second phase (line 7).

Note that as a result of this procedure, if two processes propose a value different from $\perp$ for the second phase, they propose exactly the same value. Let this value be called v^*.

The purpose of the second phase is to verify if v^* was also observed by enough other processes. After a process receives $n - f$ phase-two messages (line 7), it checks if more than f phase-two proposals are equal to v^*, and if so commits v^* (line 11). A process adopts v^* if it receives v^* in the second phase, but is unable to collect enough v^* values to decide (line 13). Finally, it is possible that a process does not receive v^* in the second phase (either because no such value was found in phase one or simply because it has received only $\perp$ in phase two); in this case the process adopts the fist value it received in phase one (line 15).

As in the Byzantine case, the main intuition is that at each round, a favorable schedule can help processes agree (i.e., adopt the same estimate), by causing them to adopt the same estimate at line 15 (e.g., by ensuring that the first INIT message they receive is from the same process). Since the schedule is random, it has a non-zero chance of being favorable at each round. Thus, almost surely, the schedule will eventually be favorable.

C.3 Proofs

We begin with a few lemmas which establish that the ROUND procedure in Algorithm 4 satisfies these properties:

Integrity If a process returns $(\cdot, v)$, then v was proposed by some process.
Strong Validity If all correct processes propose the same value v and a process returns a pair $\langle \textsc{Grade}, v' \rangle$, then $\textsc{Grade} = \textsc{Commit}$ and $v' = v$.
Consistency If any correct process returns $\langle \textsc{Commit}, v \rangle$, then no process returns $(\cdot, v' \neq v)$.
Termination If all correct processes propose, then every correct process eventually returns.

▶ **Lemma 21.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies integrity.*

Proof. We say that a value v is *valid* if it is the input value of some process. We want to show that processes only return valid values. We observe that (1) INIT messages only contain valid values (line 2), and therefore (2) ECHO messages only contain valid values or $\perp$ (lines 4–8). If a process p returns v at line 11 or 13, then p received at least one ECHO message for v, and thus v is valid by (2) above. If p returns v at line 15, then p received at least one INIT message for v, and thus v is valid by (1) above. ◀

▶ **Lemma 22.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies strong validity.*

Proof. If all processes propose the same value v, then all processes send $\langle \textsc{Init}, v \rangle$ at line 2; all processes receive at least $n/2$ $\langle \textsc{Init}, v \rangle$ messages (since $n - f = f + 1 \geq n/2$); all processes adopt v as their proposal for the second phase and send $\langle \textsc{Echo}, v \rangle$ at line 8; all processes receive $n - f = f + 1$ $\langle \textsc{Echo}, v \rangle$ and return $\langle \textsc{Commit}, v \rangle$. ◀

▶ **Lemma 23.** *If a process p sends $\langle \textsc{Echo}, v \rangle$ at line 8, then no process sends $\langle \textsc{Echo}, v' \rangle$, for any $v' \neq v$.*

Proof. Assume the lemma does not hold. Then p must have received more than $n/2$ $\langle \textsc{Init}, v \rangle$ messages, and some process p' must have received more than $n/2$ $\langle \textsc{Init}, v' \rangle$ for some $v' \neq v$. By quorum intersection, it follows that some process must have sent both an $\langle \textsc{Init}, v \rangle$ and $\langle \textsc{Init}, v' \rangle$ message. This is a contradiction, as processes only send one INIT message at line 2. ◀

▶ **Lemma 24.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies consistency.*

Proof. If process p returns $\langle \textsc{Commit}, v \rangle$, it must do so at line 11, after having received $f + 1$ $\langle \textsc{Echo}, v \rangle$ messages. Therefore, every other process p' that returns, must receive at least one $\langle \textsc{Echo}, v \rangle$ message. If p' also receives $f + 1$ $\langle \textsc{Echo}, v \rangle$ messages, then it returns $\langle \textsc{Commit}, v \rangle$. Otherwise, by Lemma 23, p' cannot receive an ECHO message for any other value $v' \neq v$, so p' returns $\langle \textsc{Adopt}, v \rangle$ at line 13. ◀

▶ **Lemma 25.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies termination.*

Proof. This follows immediately from the construction of the algorithm. Processes perform two phases; the only blocking step of each phase is waiting for $n - f$ messages (lines 3 and 9). This waiting eventually terminates, since there are at least $n - f$ correct processes. ◀

Now we can show that the consensus protocol in Algorithm 1 is correct under crash faults.

▶ **Theorem 26.** *With crash faults, Algorithm 1 satisfies validity.*

To prove the theorem, we first prove the following lemma:

▶ **Lemma 27.** *If a process proposes v to the ROUND procedure, then v is the consensus input of some process.*

Proof. We prove the result by induction on the round r. For the base case: If $r = 0$, then all processes input their consensus inputs into ROUND. For the induction case: assume the lemma holds up to $r > 0$, and consider the case of $r+1$. By the induction hypothesis and the integrity property of the ROUND procedure, any output of ROUND$(r, \cdot)$ must be the consensus input of some process. Since the input of ROUND$(r+1, \cdot)$ is the output of ROUND$(r, \cdot)$, the lemma must also hold for the case of $r + 1$. This completes the induction. ◀

Proof of Theorem 26. If a process p decides a value v, then the ROUND procedure must have output (COMMIT, v) at line 6. By Lemma 27 and the integrity property of the ROUND procedure, it follows that v is the consensus input of some process. ◀

▶ **Theorem 28.** *With crash faults, Algorithm 1 satisfies uniform agreement.*

Proof. Let r be the earliest round at which some process decides and let p be a process that decides v at round r. We will show that any other process p' that decides, must decide v.

For p to decide v at round r, ROUND must output (COMMIT, v) in that round. Thus, by the consistency property of ROUND, ROUND$(r, \cdot)$ must output $(\cdot, v)$ at all processes. If ROUND$(r, \cdot)$ outputs (COMMIT, v) for p', then p' decides v at round r (line 6). Otherwise, no other value than v can be input to ROUND$(r + 1, \cdot)$, and by the strong validity property, all processes (including p') will output (COMMIT, v) and decide v at round $r + 1$. ◀

▶ **Theorem 29.** *With crash faults, Algorithm 1 satisfies termination almost surely.*

Proof. We can describe the execution of the protocol as a Markov chain with states $0, \ldots, 2f + 1$; the system is at state i if i processes have estimate (est_i variable) equal to 0 before invoking ROUND. Due to the strong validity property of the ROUND procedure, states 0 and $2f + 1$ are absorbing states. There is a non-zero transition probability from each state, other than 0 and $2f + 1$, to each other state (we show this below). Therefore, almost surely, the system will eventually reach one of the two absorbing states and remain there. Once this happens (all processes have the same est_i variable), the strong validity property of ROUND ensures that all processes (who have not decided yet) will decide within a round.

It only remains to show that there is a non-zero transition probability from each state, other than 0 and $2f + 1$, to each other state. Consider a state $i \notin \{0, 2f + 1\}$ and a state j; there is a schedule S with non-zero probability which leads the system from i to j in one invocation of ROUND. In S, every process receives INIT messages for both 0 and 1 at line 3, and thus all processes send ECHO messages with $\perp$ at line 8. Thus, all processes return at line 15. In S, j processes receive an INIT message with value 0 first, so they adopt 0, and $2f + 1 - j$ processes receive an INIT message with value 1 first, so they adopt 1, bringing the system to state j. ◀