Gorilla: Safe Permissionless Byzantine Consensus

Youer Pu
Cornell University, Ithaca, NY, USA

Ali Farahbakhsh
Cornell University, Ithaca, NY, USA

Lorenzo Alvisi
Cornell University, Ithaca, NY, USA

Ittay Eyal
Technion, Haifa, Israel

Abstract

Nakamoto's consensus protocol works in a permissionless model and tolerates Byzantine failures, but only offers probabilistic agreement. Recently, the Sandglass protocol has shown such weaker guarantees are not a necessary consequence of a permissionless model; yet, Sandglass only tolerates benign failures, and operates in an unconventional partially synchronous model. We present Gorilla Sandglass, the first Byzantine tolerant consensus protocol to guarantee, in the same synchronous model adopted by Nakamoto, deterministic agreement and termination with probability 1 in a permissionless setting. We prove the correctness of Gorilla by mapping executions that would violate agreement or termination in Gorilla to executions in Sandglass, where we know such violations are impossible. Establishing termination proves particularly interesting, as the mapping requires reasoning about infinite executions and their probabilities.

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases Consensus, Permissionless, Blockchains, Byzantine fault tolerance, Deterministic Safety

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.31

Related Version *Full Version*: `https://arxiv.org/abs/2308.04080` [29]

Funding *Youer Pu*: Supported in part by an IC3 grant.
Lorenzo Alvisi: Supported in part by NSF Award 2106954.
Ittay Eyal: Supported in part by the Israel Science Foundation (grant No. 1641/18), an Avalanche Foundation grant, and an IC3 grant.

Acknowledgements We thank Alexandra Silva and Dexter Kozen for useful discussions about proving probabilistic termination.

1 Introduction

Nakamoto's Bitcoin [23] demonstrated that a form of consensus can be reached even if participation is permissionless. Nakamoto achieved this by introducing the cryptographic primitive Proof of Work (PoW) [10, 13] into the common synchronous Byzantine model [9]. With PoW, a process can work for a short while and probabilistically succeed in solving a puzzle. But Bitcoin only achieves a probabilistic notion of consensus: both safety and liveness fail with negligible probability.

Lewis-Pye and Roughgarden showed that deterministic and permissionless consensus cannot be achieved in a synchronous network in the presence of Byzantine failures [18]. Nonetheless, previous work (§2) has achieved deterministic safety and termination with

probability 1 under different models. Sandglass [27] assumes a benign model with a non-standard hybrid synchrony model. Momose et al. [22] guarantee termination only if the set of processes stabilizes. Malkhi et al. [20], while leveraging either authenticated channels or digital signatures, propose a solution whose correctness depends on Byzantine nodes comprising fewer than a third of the nodes in the system.

The question is whether it is possible to achieve deterministic safety and termination with probability 1 without limiting either Byzantine behavior or how nodes join and leave, and without relying on authentication.

We answer this question in the affirmative for a synchronous model (§3) with Byzantine failures. We present *Gorilla Sandglass* (or simply *Gorilla*) (§4), a consensus protocol that guarantees deterministic safety and termination with probability 1 in this standard model, which we dub *GM* (for *Gorilla Model*). Gorilla relies on a form of PoW: *Verifiable Delay Functions* (*VDFs*) [6]. We consider an ideal VDF [21] that proves a process waited for a certain amount of time and cannot be amortized. The key difference between a VDF and Nakamoto's PoW is that multiple processes can calculate multiple VDFs concurrently, but cannot, by coordinating, reduce the time to calculate a single VDF. The crux of the protocol is simple. The protocol proceeds in steps. In each step, all (correct) nodes collect VDF solutions from their peers and build new VDFs based on those. Intuitively, correct processes, which are the majority, accrue solutions faster than Byzantine nodes, and progress through the asynchronous rounds of the protocol faster. Eventually, the round inhabited by correct nodes is so far ahead of that occupied by Byzantine nodes that, no longer subject to Byzantine influence, correct nodes can safely decide.

Gorilla Sandglass adopts the general approach of Sandglass [27], in the sense that puzzle results are accrued, with each puzzle built on its predecessors. In Sandglass participants are benign and they send, in each step, a message built on previously received messages. In Gorilla, however, the Byzantine adversary is not limited to acting on step boundaries or communicating at particular times. Surprisingly, Gorilla's correctness can be reduced to the correctness of a variation of Sandglass. We perform this reduction in two steps (§5).

We first show that, for every execution of Gorilla in GM, there is a matching execution where the Byzantine processes adhere to step boundaries, in a model we call *GM+*. In the mapped execution, Byzantine processes only start calculating their VDF at the beginning of a step and only send messages at the end of a step. GM+ is a purely theoretical device, as it allows operations that cannot be implemented by actual cryptographic primitives. In particular, it allows Byzantine processes to start calculating a VDF in a step s building on any VDF computed by other Byzantine nodes that will be completed *by the end* of s, rather than by the start s, as allowed by GM (and actually feasible in reality). Nonetheless, GM+ serves as a crucial stepping stone towards proving Gorilla's correctness.

Next, we show that, given an execution in GM+ that violates correctness, there exists a corresponding execution of Sandglass in a model we call SM+. The SM+ model is similar to that of Sandglass: in both, processes are benign and propagation time is bounded for messages among correct processes and unbounded for messages to and from so-called *defective nodes*. But unlike Sandglass, in SM+ a message from a defective node can reference another message generated by another defective node during the same step (similar to how GM+ allows Byzantine nodes to calculate a VDF that builds on VDFs calculated by other Byzantine nodes in the same step).

Together, this pair of reduction steps establishes that if an execution of Gorilla in GM violates correctness with positive probability, then so does an execution of Sandglass in SM+. To conclude Gorilla's proof of correctness, all that is left to show is that Sandglass retains

deterministic safety and termination with probability 1 in the SM+ model: fortunately, the correctness proof of Sandglass [27] works almost without change (§A of [29]) in SM+. Thus, a violation of correctness in Gorilla results in a contradiction, and therefore, Gorilla is correct.

Gorilla demonstrates that it is *possible* to achieve deterministic safety and liveness with probability 1 in a permissionless Byzantine model. Yet, possible does not mean *practical*: Gorilla is not, since, like the Sandglass protocol that inspires it, it requires an exponential number of rounds to terminate. By answering the fundamental question of possibility, Gorilla ups the ante: is there a practical solution to deterministically safe permissionless consensus?

2 Related Work

Lewis-Pye et al. [18] have proven that deterministic consensus is impossible in the permissionless setting. Therefore, for at least one of safety and liveness probabilistic guarantees are inevitable. Gorilla concedes little: it manages to keep safety deterministic, and guarantees liveness with probability 1.

Several protocols [3, 4, 7, 11, 12, 16, 24] have embraced Bitcoin's permissionless participation and probabilistic safety. All rely for correctness on probabilistic mechanisms, which leave open the possibility that Byzantine nodes may overturn safety or liveness guarantees with positive probability. Gorilla avoids this peril by basing correctness on the process of accruing a deterministic number of messages.

Few proposals achieve deterministic safety in a permissionless setting [20, 22, 27]. Momose et al. [22] introduce the concept of *eventually stable participation*, akin to partial synchrony; it requires that, after an unknown global stabilization time, for each T-wide time interval $[t, t + T]$, at least half of the nodes ever awake during the interval are correct and do not leave. Gorilla guarantees progress without assuming stability in participation.

Pu et al. [27] propose Sandglass, which achieves deterministic safety but only in a benign setting. Gorilla extends Sandglass to tolerate Byzantine failures.

Malkhi et al. [20] let nodes join and leave at any time, as in Gorilla. Unlike Gorilla, however, they must rely on authenticated channels to tolerate fluctuations in the number of adversaries. Further, Byzantine nodes must be fewer than a third of active nodes, while in Gorilla they must be fewer than one half.

Several works have modeled permissionless participation [2, 17, 26].

Pass et al. [26] introduce the *sleepy participation model*, in which honest nodes are either awake or asleep. Awake nodes participate in the protocol, while asleep nodes neither participate nor relay messages. Byzantine nodes are always awake, but the scheduler can adaptively turn an honest node Byzantine, as long as Byzantine nodes remain a minority of awake nodes. Gorilla similarly assumes that correct and Byzantine nodes can join and leave at any time, as long as a majority of active nodes are correct. Unlike the sleepy model, however, Gorilla requires no public key infrastructure, and, unlike sleepy consensus, guarantees deterministic safety.

Unlike Gorilla, Lewis-Pye et al. [17] do not offer a consensus protocol, but rather focus on introducing *resource pools*, an abstraction that aims to capture resources used to establish identity in permissionless systems, *e.g.*, computational power through PoW and fiscal power through Proof of Stake (PoS).

Aspnes et al. [2] explore consensus in an asynchronous benign model where an unbounded number of nodes can join and leave, but where at least one node is required to live forever, or until termination. Gorilla instead assumes a synchronous model, tolerates Byzantine failures, and allows any node to join and leave, as long as a majority of active nodes is correct.

Verifiable Delay Functions (VDFs) [6] have been leveraged as a resource against Byzantine adversaries in various works [8, 15, 19, 30], specifically to defend PoS systems from attacks where participants can go back in time and mine blocks. Gorilla leverages VDFs to rate-limit the ability of Byzantine nodes to create valid messages.

3 Model

The system is comprised of an infinite set of nodes $\{p_1, p_2, \dots\}$. Time progresses in discrete ticks $0, 1, 2, 3, \dots$ In each tick, a subset of the nodes is *active*; the rest are *inactive*. The upper bound on active nodes in any tick, necessary to the safety of Nakamoto's permissionless consensus [25], is $\mathcal{N}$, and there is at least one active node in every tick. Starting from tick 0, every K ticks are grouped into a step: each step i consists of ticks $iK, iK+1, \dots, iK+K-1$.

A Verifiable Delay Function (VDF) is a function whose calculation requires completing a given number of sequential steps. Thus, evaluating a VDF requires the evaluator to spend a certain amount of time in the process. Specifically, we require the evaluation of a single VDF to take K ticks. We refer to the intermediate random values that this evaluation produces at the end of each of the K ticks as the *units of the VDF evaluation* (or, more succinctly, the *units of the VDF*). We denote the i-th unit of evaluating the VDF of some input γ by vdf_γ^i; we denote the final result (*i.e.,* vdf_γ^K) by vdf_γ, or, when there is no ambiguity, by *vdf*.

We model the calculation of VDFs with the help of an *oracle* Ω. Nodes use Ω both to iteratively obtain the units of a VDF and to verify whether a given value is the *vdf* of a given input. In particular, Ω provides the following API:

Get(γ, vdf_γ^i): returns $vdf_\gamma^{(i+1)}$. By convention, invoking Get$(\gamma, \perp)$ returns vdf_γ^1. The oracle remembers how it responded to a Get query – so that, even though the units of a VDF are random values, identical queries produce identical responses. Ω accepts at most one call to Get() in any tick from each node.

Verify(vdf, γ): returns True iff $vdf = vdf_\gamma^K$. Ω accepts any number of calls to Verify() in any tick from any node.

If Get$(\gamma, \perp)$ is called at tick t and step s, we say the VDF calculation for γ *starts* at tick t and step s. Similarly, the VDF calculation for γ *finishes* at tick t and step s if Get$(\gamma, vdf_\gamma^{K-1})$ is called at tick t and step s.

In each tick, an active node receives a non-negative number of messages, updates its variables – potentially including calls to the oracle – and then communicates with others using a synchronous broadcast network. The network allows each active node to *broadcast* and *receive* unauthenticated messages. Node p_i invokes $Broadcast_i(m)$ to broadcast a message m, and receives broadcast messages from other nodes (and itself) by invoking $Receive_i$. The network neither generates nor duplicates messages and ensures that if a node receives a message m in tick t, then m is broadcast in tick $(t-1)$. The network propagation time is negligible compared to a tick, *i.e.*, to the time necessary to calculate a unit of a VDF. By executing the command $Receive_i$, a newly joining node p_i receives all messages broadcast by correct nodes prior to its activation. Nodes whose network connections with other nodes are asynchronous can be modeled as Byzantine, as Byzantine nodes can deliberately or unintentionally delay messages sent from or to them. Therefore, Gorilla also tolerates asynchrony, as long as the nodes that communicate asynchronously are a minority.

Correct nodes do not deviate from their specification and constitute a majority of active nodes at each tick. Correct nodes always join at the beginning of a step and leave when a step ends. Hence, a correct node is active from the first to the last tick of a step. The remaining nodes are *Byzantine* and can suffer from arbitrary failures. Byzantine nodes can join and leave at any tick.

All nodes are initialized with a value $v_i \in \{0, 1\}$ upon joining the system. An active node p_i decides by calling $Decide_i(v)$ for some value v. A protocol solves the consensus problem if it guarantees the following properties [9]:

▶ **Definition 1** (Agreement). *If a correct node decides a value v, then no correct node decides a value other than v.*

▶ **Definition 2** (Validity). *If all nodes that ever join the system have initial value v and there are no Byzantine nodes, then no correct node decides $v' \neq v$.*

▶ **Definition 3** (Termination). *Every correct node that remains active eventually decides.*

4 Gorilla

Gorilla borrows its general structure from Sandglass (see Algorithm 1) [27]. Executions proceed in asynchronous rounds (even though, unlike Sandglass, Gorilla assumes a standard synchronous model of communication between all nodes). Upon receiving a threshold of valid messages for the current round, nodes progress to the next round; if all the messages received by a correct node propose the same value v for sufficiently many consecutive rounds, the node decides v. The number of active nodes is bounded by $\mathcal{N}$ but otherwise unknown. Within this bound, it can fluctuate arbitrarily, but both safety and liveness depend on the correctness of a majority of nodes.

The key aspects of the protocol can be summarized as follows:

Ticks, steps and VDF Each valid message must contain a *vdf*. A correct node takes a full step, *i.e.*, K consecutive ticks, to individually calculate a *vdf*, and at the end of the step sends a valid message that contains the *vdf*. Byzantine nodes may instead share among themselves the work required to finish the K units of a VDF calculation; even so, it still takes K distinct ticks for Byzantine nodes to compute a *vdf*. Requiring valid messages to carry a *vdf* limits Byzantine nodes to sending messages at the same rate as correct nodes; this ensures that, on average across all steps, the correct majority sends at least one more valid message than the minority of nodes that are Byzantine.

Choosing a threshold A node proceeds to round r if it receives at least $\mathcal{T} = \lceil \frac{\mathcal{N}^2}{2} \rceil$ messages for round $r - 1$. Even though setting such a threshold does not prevent Byzantine nodes from advancing from round to round, it nonetheless gives the correct nodes an edge in the pace of such progress, since they constitute a majority.

Exchanging messages In each step of the protocol, a node in any round r – based on the messages it has received so far – searches for the largest round $r_{max} \geq r$ for which it has accrued $\mathcal{T}$ messages. It then broadcasts a message for the next round. The message includes the node's current proposed value v, the *vdf*, and four other attributes discussed below: the message's *coffer*, a nonce, as well as v's *priority* and *unanimity counter*.

Keeping history Nodes can join the system at any time. To help a joining node catch up, every message broadcast by a node p in round r includes a *message coffer* that contains: (i) messages from round $r - 1$ received by p to advance to round r; (ii) recursively, messages included in those messages' coffers; and (iii) messages received by p for round r.

Nonce By making it possible to distinguish between messages that are generated from the same coffer, nonces allow correct nodes to broadcast multiple valid messages during a round while, at the same time, preventing Byzantine nodes from reusing the same *vdf* to send multiple valid messages based on a given message coffer.

Priority and unanimity counter If a node p only receives the value v from a majority for a sufficient number of consecutive rounds, it decides v. To guarantee the safety of this decision, p assigns a *priority* to the value v that it proposes. This priority is incremented once v is unanimously proposed for a long stretch of consecutive rounds. To record the length of this stretch, each node computes it upon entering a round r, and includes it as the *unanimity counter* in the messages it sends for round r. If a node collects more than one value in a round r, it chooses the one with the highest priority, and proposes it for round $r + 1$. In case of a tie, it uses *vdf* as a source of randomness to choose one of the values randomly. Since *vdf* is a random number calculated based on the message coffer and a nonce (lines 13-15), a Byzantine node is unable to deliberately pick an input to VDF to deterministically get the desired value.

Message internal consistency and validity A message m is *internally consistent* if the attributes carried by m can be generated by following Gorilla correctly based on the message coffer carried in m. We denote the *vdf* in m by vdf_m.

A message m is *valid* (and thus isValid(m) returns true), if (i) vdf_m can be verified by the message coffer and the nonce of m; (ii) m is internally consistent; and (iii) for any message m' in m's coffer, m' is also valid. Otherwise, m is invalid.

In addition to demonstrating variable initialization, Algorithm 1 presents the algorithm each node p_i runs at each step. Each node p_i starts every step by adding all valid messages, in addition to the messages in their coffers, to the set Rec_i (lines 4-6).

Iterating over Rec_i, node p_i computes the largest round r_{max} for which it has received at least $\mathcal{T}$ messages, and updates its current round to $r_{max} + 1$ (line 8) if the condition in line 7 holds. Once in a new round, p_i does the following: (i) resets its message coffer M and adds to it the messages it has received from the previous round – alongside the messages in *their* coffers (lines 9-11); (ii) picks a nonce and calculates a *vdf* based on its coffer and the nonce (lines 13-15); (iii) chooses its proposal value (lines 16 -20); it chooses the proposal with the highest priority among the previous round messages in its coffer; in case of a tie, it chooses a random number utilizing the randomness in *vdf*; (iv) determines the priority and the unanimity counter for the messages it will broadcast in the current round (lines 21-25); and finally (v) the node decides v if v's priority is high enough (lines 26-27). If p_i does not enter a new round, it starts to create a message nonetheless: it adds to the message's coffer all messages received for the current round (line 29), and calculates a *vdf* with the new message coffer and a different nonce as the input (lines 30-32), so that the message is unique. Regardless of whether it enters a round or not, p_i ends every step by broadcasting the message it has created (line 33).

Comparing Sandglass and Gorilla

Gorilla retains the structure of Sandglass, adding the requirement that valid messages must include a *vdf* and a nonce. The differences between the protocols are highlighted in orange in Algorithm 1: (i) *vdf* is calculated for each message sent (lines 13-15,30-32), (ii) received messages are checked to see if they are valid (line 5); (iii) *vdf* is used as the source of randomness (line 20) where the protocol requires choosing a value randomly.

These additions are critical to handling Byzantine faults. Both Gorilla and Sandglass rely on correct (respectively, good) nodes sending the majority of unique messages during an execution. In Sandglass, where defective nodes are benign, this property simply follows from requiring correct nodes to be a majority in each step; not so in Gorilla, where faulty nodes can be Byzantine. Requiring valid message in Gorilla to carry a *vdf* preserves correctness by effectively rate-limiting Byzantine nodes' ability to create valid messages.

Algorithm 1 Gorilla: Code for node p_i. The orange text highlights where Gorilla departs from Sandglass.

1: **procedure** INIT($input_i$)
2: $v_i \leftarrow input_i$; $priority_i \leftarrow 0$; $uCounter_i \leftarrow 0$; $r_i = 1$; $M_i = \emptyset$; $Rec_i = \emptyset$;
3: **procedure** STEP
4: **for all** $m = (\cdot, \cdot, \cdot, \cdot, \cdot, M)$ received by p_i **do**
5: **if** isValid(m) **then**
6: $Rec_i \leftarrow Rec_i \cup \{m\} \cup M$
7: **if** $\max_{|Rec_i(r)| \geq \mathcal{T}}(r) \geq r_i$ **then**
8: $r_i = \max_{|Rec_i(r)| \geq \mathcal{T}}(r) + 1$
9: $M_i = \emptyset$
10: **for all** $m = (\cdot, r_i - 1, \cdot, \cdot, \cdot, M) \in Rec_i(r_i - 1)$ **do**
11: $M_i \leftarrow M_i \cup \{m\} \cup M$
12: $M_i \leftarrow M_i \cup Rec_i(r_i)$
13: $vdf \leftarrow \perp$; $nonce \leftarrow$ a new arbitrary value
14: **for** $j : 1..k$ **do**
15: $vdf \leftarrow Get((M_i, nonce), vdf)$
16: Let C be the multi-set of messages in $M_i(r_i - 1)$ with the largest priority.
17: **if** all messages in C have the same value v **then**
18: $v_i \leftarrow v$
19: **else**
20: $v_i \leftarrow vdf \mod 2$
21: **if** all messages in $M_i(r_i - 1)$ have the same value v_i **then**
22: $uCounter_i \leftarrow 1 + \min\{uCounter | (\cdot, r_i - 1, v_i, \cdot, uCounter, \cdot) \in M_i(r_i - 1)\}$
23: **else**
24: $uCounter_i \leftarrow 0$
25: $priority_i \leftarrow \max(0, \lfloor \frac{uCounter_i}{\mathcal{T}} \rfloor - 5)$
26: **if** $priority_i \geq 6\mathcal{T} + 4$ **then**
27: Decide$_i(v_i)$
28: **else**
29: $M_i \leftarrow M_i \cup Rec_i(r_i)$
30: $vdf \leftarrow \perp$; $nonce \leftarrow$ a new arbitrary value
31: **for** $j : 1..k$ **do**
32: $vdf = Get((M_i, nonce), vdf)$
33: broadcast $(r_i, v_i, priority_i, uCounter_i, M_i, nonce, vdf)$

Given their differences in both failure model and timing assumptions, it is perhaps surprising that so little needs to change when moving from Sandglass to Gorilla. After all, Sandglass assumes a model where failures are benign and a hybrid synchronous model of communication [28]; Gorilla instead assumes a Byzantine failure model, and a synchronous network model (§3). Note, however, that although Sandglass assumes benign failures, its hybrid communication model implicitly accounts for Byzantine nodes strategically choosing the timing for receiving and sending messages to correct nodes: Gorilla can then simply inherit from Sandglass the mechanisms for tolerating such behaviors.

5 Correctness

Despite the similarlity between the Gorilla and Sandglass protocols, proving Gorilla's correctness directly is challenging. Unlike Sandglass, Byzantine nodes can act between step boundaries, interleave VDF computations instead of producing one VDF (and hence one message) at the time, etc. To overcome this complexity, our approach is to leverage as much as possible Sandglass's proof of correctness.

Our battle plan was to first map executions of Gorilla to executions of Sandglass. Then we intended to proceed by contradiction: assume that a correctness guarantee is violated in Gorilla, and map this violation to Sandglass; since correctness violations are not possible in Sandglass [27], we could then conclude that neither they can be in Gorilla.

The best laid plans often go awry, and, as we discuss below, ours was no exception – but we were able to nonetheless retain the conceptual simplicity of our initial approach.

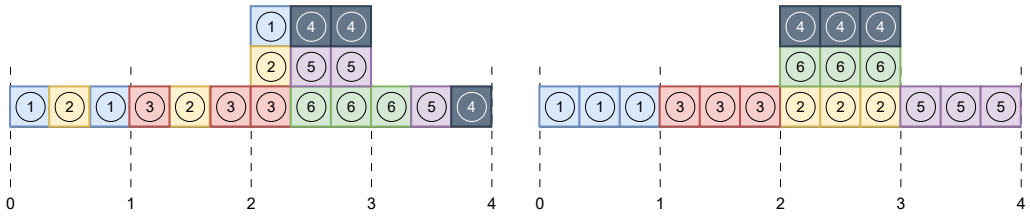
5.1 The Main Story, and How it Fails

The mapping from Gorilla to Sandglass must satisfy certain *well-formedness* and *equivalence* conditions. The former specify how to map a Gorilla execution into one that satisfies the Sandglass model (SM) and follows the Sandglass protocol; the latter allow us to map violations from Gorilla to Sandglass, *i.e.*, they preserve certain properties of the behavior of *correct* nodes in Gorilla and reinterpret them as the behavior of *good* nodes in Sandglass.

Well-formedness requires mapping correct nodes to good nodes, and Byzantine nodes to defective nodes, while respecting model constraints (*e.g.*, at each step defective nodes should be fewer than good nodes). The first half of this mapping is easy: except for calculating a VDF, correct nodes in GM are not doing anything different than good nodes in SM. Thus, mapping a step in GM to a step in SM yields a straightforward connection between correct and good nodes. The second half, however, is trickier. Defective nodes in SM can suffer from benign faults like omission and crashing, but these fall short of fully capturing Byzantine behavior in GM. In particular, Byzantine nodes, even when sending valid messages, can violate the timing constraints that Gorilla places on a node's actions, *e.g.*, by splitting the calculation of a single VDF into multiple steps. Thus, before a Gorilla execution can be mapped to a Sandglass execution, Byzantine nodes' actions must be brought to conform to step boundaries and not spill across steps. After tidying things up this way, it must become possible to map the faulty actions of the Byzantine nodes to a combination of crashes, omissions, and network delays, *i.e.*, to the faults and anomalies that SM allows.

Equivalence in turn requires that, when mapping executions from Gorilla to Sandglass, a correct node and its corresponding good node send and receive in every step messages that allow them to update their proposed value, round number, priority, and unanimity counter in the same way. Since messages play the same role in both protocols, this is sufficient for good nodes in Sandglass to decide identically to the corresponding correct nodes in Gorilla.

Our plan to realize this logical mapping involved splitting it into two concrete, intermediate mappings: a first mapping from an initial Gorilla execution to an intermediate Gorilla execution in which Byzantine actions conform to step boundaries; and a second mapping from that intermediate execution to a Sandglass execution. We require all of our well-formedness and equivalence conditions to hold throughout these mappings: (*i*) model constraints must be always respected, (*ii*) correct nodes in the intermediate execution send and receive the equivalent (indeed, the same!) messages as their counterparts in the initial execution, at the same steps, and (*iii*) good nodes in the final execution send and receive equivalent messages as their correct counterparts in the intermediate execution, at the same steps.



(a) The counterexample. **(b)** The solution enabled by peeking.

Figure 1 An execution that cannot be reorganized in GM (a), and how peeking solves the problem in GM+ (b).

Unfortunately, *well-formedness and equivalence cannot be satisfied by the first mapping.* To see why, consider Figure 1a. Here, each square represents a VDF unit calculated by a Byzantine node for a specific input, denoted by a unique color. Numbered circles represent the corresponding messages, *e.g.*, the VDF units containing ① are associated with message ①. Each VDF calculation takes three ticks, and a step comprises three ticks. The numbered dashed lines indicate the steps, *i.e.*, the three ticks between lines i and $i+1$ belong to step i. Assume that, to maintain a majority of correct nodes in the system, the maximum allowable number of Byzantine nodes in the four steps shown in the figure are, respectively, 1, 1, 3, and 1. Moreover, assume that messages ④, ⑤, and ⑥ all include in their coffers messages ①, ②, and ③. Finally, assume that messages ④, ⑤, and ⑥ are sent to correct nodes at the start of Step 4. Since the actions of Byzantine nodes in Figure 1a do not conform to step boundaries, the first mapping should be able to organize them in a way that ensures that (i) correct nodes receive messages ④, ⑤, and ⑥ at the beginning of Step 4, and (ii) each of these messages in turn includes messages ①, ②, and ③. Thus, the calculation of the VDFs for messages ①, ②, and ③ must be completed before those for ④, ⑤, and ⑥ can start. Now, since steps 0 and 1 include only one Byzantine node, they can only accommodate one VDF, *i.e.*, only one VDF can be calculated in each of steps 0 and 1. Without loss of generality, let those VDFs be ① and ③, respectively. VDF ② must still complete before messages ④, ⑤, and ⑥: thus, it has to be placed in Step 2. Note that, although Step 2 could accommodate two more Byzantine VDFs at Step 2, they cannot be placed there, since the completion of VDF ② must precede the start of the calculation of VDFs ④, ⑤, and ⑥: the earliest step where they can start is Step 3. However, it is impossible to accommodate all three there, since in Step 3 there is a single Byzantine node.

Our first attempt at mapping executions from Gorilla to Sandglass has thus failed. Fortunately, though, it is possible to retain the strategy that underlies it and overcome the above counterexample without weakening our well-formedness and equivalence conditions. Instead, we proceed to weaken the model in which we operate, by giving Byzantine nodes extra power.

5.2 A New Beginning

The first step in our two-step process for mapping a Gorilla execution η_G into a Sanglass execution η_S is to reorganize the actions taken by Byzantine nodes in η_G: we want to map η_G to an execution where Byzantine nodes join the system and receive valid messages at the beginning of a step (by the first tick) and broadcast valid messages and leave the system at the step's end (at its K-th tick). Since, as explained in Section 5.1, satisfying all of these requirements is not possible, we extend GM to a new model.

We need some way to calculate a VDF on an input that includes the final result of VDF calculations that are still in progress. To achieve this, we extend the oracle's API to allow Byzantine nodes to *peek* at those future outcomes. By issuing the oracle a *peek* query, Byzantine nodes active in any step s can learn the result of a VDF computed by Byzantine nodes finishing at step s before its calculation has ended.

We thus introduce GM+, a model that extends GM by having a new oracle, Ω^+, that supports one additional method:

Peek(γ): immediately returns vdf_γ.

In any tick, a Byzantine node in GM+ can call Peek() multiple times, with different inputs. However, Byzantine nodes can only call Peek subject to two conditions:

- A Byzantine node can peek in step s at vdf_γ only if Byzantine nodes commit to finish the VDF calculation for input γ within s; and
- a Byzantine node does not peek at vdf_γ, where $\gamma = (M, nonce)$, if M in turn contains some VDF result v obtained by peeking, and the calculation of v has yet to finish in this tick.

Note that these restrictions only limit the *additional* powers that GM+ grants the adversary: in GM+, Byzantine nodes remain strictly stronger than in GM.

With this new model, taking a detour, we first map an execution of Gorilla in GM to an execution of Gorilla in GM+, in which Byzantine behavior is reorganized with the addition of peeking. Hence follows the first lemma of our scaffolding: the existence of the first mapping.

▶ **Definition 4.** *Consider an execution η_G in GM and an execution η_G^+ in GM+. We say η_G and η_G^+ are* equivalent *iff the following conditions are satisfied:*

Reorg-1 *For every correct node p in η_G, there exists a correct node p^+ in η_G^+, such that p and p^+ (i) join and leave the system at the same ticks in the same steps and (ii) receive and send the same messages at the same ticks in the same steps.*

Reorg–2 *Each Byzantine node in η_G^+ (i) joins at the first tick of a step and leaves after the last tick of that step; (ii) receives messages at the first tick of a step and sends messages at the last tick of that step; and (iii) sends and receives only valid messages.*

Reorg-3 *If in η_G a Byzantine node sends a valid message m at a tick in step s, then in η_G^+ a Byzantine node sends m at a tick in some step $s' \leq s$.*

▶ **Lemma 5.** *There exists a mapping* Reorg *that maps an execution η_G in GM to an execution η_G^+ in GM+, denoted $\eta_G^+ = \textsc{Reorg}(\eta_G)$, such that η_G is equivalent to η_G^+.*

While peeking solves the challenge with reorganizing Byzantine behavior, it complicates our second mapping. The ability to peek granted to Byzantine nodes in GM+ has no equivalent in Sandglass – it simply cannot be reduced to the effects of network delays or to the behavior of defective nodes. Therefore, we weaken SM so that defective nodes can benefit from a capability equivalent to peeking.

We do so by introducing SM+, an extension of SM that is identical to SM, except for the following change: defective nodes at step s can receive any message m sent by a defective node no later than s – as opposed to $(s-1)$ in SM – as long as m does not contain in its coffer a message that is sent at s. Note that allowing defective nodes to receive in a given step a message m sent by defective nodes within that very step maps to allowing Byzantine nodes to peek at a message whose *vdf* will be finished by Byzantine nodes within the same step; and the constraint that m shouldn't contain in its coffer other messages sent in the same step, maps to the constraint that Byzantine nodes cannot peek at messages whose coffer also contains a peek result from the same step.

One might rightfully ask: was not the plan to leverage the correctness of Sandglass in SM? Indeed, but fortunately, *Sandglass still guarantees deterministic agreement and termination with probability 1 under the SM+ model* (§A.3 of [29]). Thus, it is suitable to map a Gorilla execution in GM+ to a Sandglass execution in SM+, and orient our proof by contradiction with respect to the correctness of Sandglass in SM+.

Formally, we specify our second mapping as follows.

▶ **Definition 6.** *Given a message m in the Gorilla protocol, the mapping* MAPM *produces a message in the Sandglass protocol as follows*

1. *Omit the vdf and the nonce from m.*
2. *Let p_i be the node that sends m. Include p_i as a field in m.*
3. *If m is the j-th message sent by p_i, add a field $uid = j$ to m.*
4. *Repeat the steps above for all of the messages in m's coffer.*

Denote the result by $\hat{m} = \text{MAPM}(m)$. We say m and $\hat{m}$ are equivalent*. Furthermore, with a slight abuse of notation, we apply* MAPM *to a set of messages as well,* i.e.*, if $\mathcal{M}$ is a set of messages, and we map each message $m \in \mathcal{M}$, we obtain the message set $\text{MAPM}(\mathcal{M})$.*

▶ **Definition 7.** *Consider an execution η_G^+ in GM+ and an execution η_S^+ in SM+. We say η_G^+ and η_S^+ are* equivalent *iff the following conditions are satisfied:*

1. *The nodes in η_G^+ are in a one-to-one correspondence with the nodes in η_S^+. For every node p in η_G^+, we denote the corresponding node in η_S^+ with $\hat{p}$.*
2. *Nodes p and $\hat{p}$ join and leave at the same steps in η_G^+ and η_S^+, respectively. Furthermore, their initial values are the same.*
3. *If p is a Byzantine node, then $\hat{p}$ is defective in SM+; otherwise, $\hat{p}$ is a good node in SM+.*
4. *$\hat{p}$ sends $\hat{m}$ at step s in η_S^+, iff p generates a message m in η_G^+ at step s. Note that in η_G^+, correct nodes send their messages to all as soon as they are generated, while Byzantine nodes may only send their messages to a subset of nodes once their messages are generated.*
5. *$\hat{p}$ receives $\hat{m}$ at step s in η_S^+, iff p receives m at step s in η_G^+.*

▶ **Lemma 8.** *Consider any execution η_G in GM, and an execution η_G^+ in GM+ equivalent to η_G. There exists a mapping* INTERPRET *that maps η_G^+ to an execution η_S^+ in SM+, denoted as $\eta_S^+ = \text{INTERPRET}(\eta_G^+)$, such that η_S^+ is equivalent to η_G^+.*

Finally, for our proof by contradiction to work, we have to show that Sandglass is correct in SM+. The proof is deferred to §A.3 of [29].

▶ **Theorem 9.** *Sandglass satisfies agreement and validity deterministically and termination with probability 1 in SM+.*

5.3 Safety

We prove that Gorilla satisfies Validity and Agreement. The proofs follow the same pattern: assume a violation exists in some execution η_G of Gorilla running in GM; map that execution to $\eta_G^+ = \text{REORG}(\eta_G)$ in GM+; then, map η_G^+ again to $\eta_S^+ = \text{REORG}(\eta_G^+)$ in SM+; and, finally, rely on the fact that these mappings ensure that correct nodes in η_G and good nodes in η_S^+ reach the same decisions in the same steps to drive a contradiction.

This approach is made rigorous in following lemmas, proved in §B.2 of [29].

▶ **Lemma 10.** *Consider an arbitrary Gorilla execution η_G, and $\eta_G^+ = \text{REORG}(\eta_G)$. If a correct node p decides a value v at step s in η_G, then p's corresponding node p^+ decides v at step s in η_G^+.*

▶ **Lemma 11.** *Consider any execution η_G in GM. If an execution $\eta_G^+ = \text{REORG}(\eta_G)$ in GM+ and an execution η_S^+ in SM+ are equivalent, then the following statements hold:*

1. *If a correct node p decides a value v at step s in η_G^+, then $\hat{p}$ decides v at step s in η_S^+.*
2. *Consider the first message $m = (r, v, priority, uCounter, M, nonce, vdf)$ that p generates for round r. Let the step when m is generated be s. If $uCounter$ is 0, then $\hat{p}$ randomly chooses value v as the proposal value at step s in η_S^+.*

We can now state and prove the safety guarantees.

▶ **Theorem 12.** *Gorilla satisfies agreement in GM.*

Proof. By contradiction, assume that there exists a Gorilla execution η_G in GM that violates agreement. This means that there exist two correct nodes p_1 and p_2, two steps s_1 and s_2, and two values $v_1 \neq v_2$ such that p_1 decides v_1 at s_1 and p_2 decides v_2 at s_2. Consider $\eta_G^+ = \text{REORG}(\eta_G)$. According to Lemma 10, p_1^+ decides v_1 at s_1 and p_2^+ decides v_2 at s_2, in η_G^+. Now, consider $\eta_S^+ = \text{INTERPRET}(\eta_G^+)$. According to Lemma 11, $\hat{p}_1^+$ decides v_1 at s_1 and $\hat{p}_2^+$ decides v_2 at s_2, in η_S^+. However, this contradicts the fact Sandglass satisfies agreement in SM+ (Theorem 9). Therefore, Gorilla satisfies agreement in GM. ◀

▶ **Theorem 13.** *Gorilla satisfies validity in GM.*

Proof. By contradiction, assume that there exists a Gorilla execution η_G, such that (i) all nodes that ever join the system have initial value v; (ii) there are no Byzantine nodes; and (iii) a correct node p decides $v' \neq v$.

Since GM+ is an extension of GM, η_G conforms to GM+. According to Definition 4, $\eta_G^+ = \eta_G$ in GM+ is trivially equivalent to η_G. Consider $\eta_S^+ = \text{INTERPRET}(\eta_G^+)$.

By the construction of the INTERPRET mapping (in Lemma 8), good nodes in η_S^+ have the same initial values as their corresponding correct nodes in η_G. Furthermore, since there are no Byzantine nodes in η_G^+, there are no defective nodes in η_S^+ by Definition 7. Therefore, by Validity of Sandglass in SM+ (Theorem 9), no good node decides $v' \neq v$. However, by Lemma 10 and Lemma 11, p decides $v' \neq v$, which leads to a contradiction. Therefore, Gorilla satisfies validity in GM. ◀

5.4 Liveness

Similar to the safety proof, the liveness proof proceeds by contradiction: it starts with a liveness violation in Gorilla, and maps it to a liveness violation in Sandglass.

Formalizing the notion of violating termination with probability 1 requires specifying the probability distribution used to characterize the probability of termination. To do so, we first have to fix all sources of non-determinism [1, 5, 14]. For our purposes, non-determinism in GM and GM+ stems from correct nodes, Byzantine nodes and their behavior; in SM+, it stems from good nodes, defective nodes and the scheduler.

For correct, good, and defective nodes, non-determinism arises from the joining/leaving schedule and the initial value of each joining node. For Byzantine nodes in GM and GM+, fixing non-determinism means fixing their action *strategy* according to the current history of an execution. Similarly, fixing the scheduler's non-determinism means specifying the timing of message deliveries and the occurrence of benign failures, based on the current history. We, therefore, define non-determinism formally in terms of an environment and a strategy.

To this end, we introduce the notion of a *message history*, and define what it means for a set of messages exchanged in a given step to be *compatible* with the message history that precedes them.

▶ **Definition 14.** *For any given execution in GM and GM+ (resp., SM+), and any step s, the* message history up to s*, $\mathcal{MH}_s$, is the set of $\langle m, p, s' \rangle$ triples such that p is a correct node (resp., good node) and p receives m at $s' \leq s$.*

▶ **Definition 15.** *We say a set $\mathcal{MP}_{s+1}$ of $\langle m, p, s+1 \rangle$ triples is* compatible *with a message history up to s, $\mathcal{MH}_s$, if there exists an execution such that for any $\langle m, p, s+1 \rangle \in \mathcal{MP}_{s+1}$, the correct node (resp., good node) p receives m at step $(s+1)$.*

▶ **Definition 16.** *An environment $\mathcal{E}$ in GM and GM+ (resp., SM+) is a fixed joining/leaving schedule and fixed initial value schedule for correct nodes (resp., good and defective nodes).*

▶ **Definition 17.** *Given an environment $\mathcal{E}$, a strategy $\Theta_{\mathcal{E}}$ for the Byzantine nodes (resp., scheduler) in GM and GM+ (resp., SM+) is a function that takes the message history $\mathcal{MH}_s$ up to a given step s as the input, and outputs a set $\mathcal{MP}_{s+1}$ that is compatible with $\mathcal{MH}_s$.*

Before proceeding, there is one additional point to address. The most general way of eliminating non-determinism is to introduce randomness through a fixed probability distribution over the available options. However, the following lemma, proved in §B.3 of [29], establishes that Byzantine nodes do not benefit from employing such a randomized strategy.

▶ **Lemma 18.** *For any environment $\mathcal{E}$, if there exists a* randomized *Byzantine strategy for Gorilla that achieves a positive non-termination probability, then there exists a* deterministic *Byzantine strategy for Gorilla that achieves a positive non-termination probability.*

Since the output *vdf* of a call to the VDF oracle is a random number, the ($vdf \mod 2$) operation in line 20 of Gorilla is equivalent to tossing an unbiased coin. Given a strategy $\Theta_{\mathcal{E}}$,[1] the nodes might observe different coin tosses as the execution proceeds; thus, the strategy specifies the action of the Byzantine nodes for all possible coin toss outcomes. The scheduler's strategy in SM+ is similarly specified for all coin toss outcomes. Therefore, once a strategy is determined, it admits a *set* of different executions based on the coin toss outcomes; we denote it by H_Θ. Specifically, a strategy determines an action for each outcome of any coin toss.

Given a strategy Θ, we can define a probability distribution P_{H_Θ} over H_Θ. For each execution $\eta \in H_\Theta$, there exists a unique string of zeros and ones, representing the coin tosses observed during η. Denote this bijective correspondence by $\textsc{Coins} : H_\Theta \to \{0,1\}^* \cup \{0,1\}^\infty$, and the probability distribution on the coin toss strings in $\textsc{Coins}(H_\Theta)$ by $\tilde{P}_{H_\Theta}$. For every event $E \subset H_\Theta$, if $\textsc{Coins}(E)$ is measurable in $\textsc{Coins}(H_\Theta)$, then $\tilde{P}_{H_\Theta}(\textsc{Coins}(E))$ is well-defined; thus, $P_{H_\Theta}(E)$ is also well-defined and $P_{H_\Theta}(E) = \tilde{P}_{H_\Theta}(\textsc{Coins}(E))$. We denote P_{H_Θ} as the probability distribution induced over H_Θ by its coin tosses.

Equipped with these definitions, we can formally define termination with probability 1.

▶ **Definition 19.** *The Gorilla protocol terminates with probability 1 iff for every environment $\mathcal{E}$ and every Byzantine strategy Θ based on $\mathcal{E}$, the probability of the termination event T in H_Θ,* i.e.*, $P_{H_\Theta}(T)$, is equal to 1.*

This definition gives us the recipe for proving by contradiction that Gorilla terminates with probability 1. We first assume there exists a Byzantine strategy Θ that achieves a non-zero non-termination probability, and map this strategy through the Reorg and Interpret mappings to a scheduler strategy Λ that achieves a non-zero non-termination probability in SM+. However, Λ cannot exist, as the Sandglass protocol terminates with probability 1 in SM+ (Theorem 9).

[1] When it is clear from the context, we will omit the environment from the subscript of the strategy.

▶ **Lemma 20.** *If there exists an environment $\mathcal{E}$ and a Byzantine strategy $\Theta_{\mathcal{E}}$ in GM that achieves a positive non-termination probability, then there exists an environment $\mathcal{E}'$ and a Byzantine strategy $\Psi_{\mathcal{E}'}$ in GM+ that also achieves a positive non-termination probability.*

Proof. Assume there exist an environment $\mathcal{E}$ and a Byzantine strategy $\Theta_{\mathcal{E}}$ in GM that achieves a positive non-termination probability. Consider the Reorg mapping. Since, according to Lemma 5, the joining/leaving and initial value schedules for correct nodes remain untouched by the Reorg mapping, we just set $\mathcal{E}' = \mathcal{E}$. In the rest of the proof, we omit the environments for brevity.

We now show that the strategy Ψ exists, and is in fact the same as Θ. For brevity, let R_Θ denote $\textsc{Reorg}(H_\Theta)$, and consider any execution η in H_Θ. By Lemma 5, correct nodes in η receive the same messages, at the same steps, as the correct nodes in $\textsc{Reorg}(\eta)$ and, moreover, the coin results in η are exactly the same as the ones in $\textsc{Reorg}(\eta)$. Thus, the message history of correct nodes up to any step s in η is the same as the message history of correct nodes up to the same step in $\textsc{Reorg}(\eta)$. In addition, because $\textsc{Reorg}(\eta)$ is a GM+ execution, compatibility is trivially satisfied. Thus, we conclude that Byzantine nodes in R_Θ follow the same strategy as in Θ, conforming to the same coin toss process. Let us denote this strategy with Ψ.

Note that according to Lemma 10, whenever a correct node decides at some step s in η, its corresponding correct node in $\textsc{Reorg}(\eta)$ decides the same value at the same step. Therefore, the set of non-terminating executions in H_Θ are mapped to the set of non-terminating executions in R_Θ in a bijective manner. Let us denote these sets as NT_H and NT_R, respectively. Since the same coin toss process induces probability distributions P_{H_Θ} and P_{R_Θ} on H_Θ and R_Θ, respectively, we conclude that $P_{H_\Theta}(NT_H) = P_{R_\Theta}(NT_R)$. Therefore, since $P_{H_\Theta}(NT_H) > 0$ by assumption, this concludes our proof, as we have shown the existence of a strategy Ψ in GM+ that achieves a positive non-termination probability. ◀

A similar lemma applies to the second mapping. We prove it in §B.3 of [29].

▶ **Lemma 21.** *If there exists an environment $\mathcal{E}$ and a strategy Ψ for Byzantine nodes in GM+ that achieves a positive non-termination probability, then there exists an environment $\mathcal{E}'$ and a scheduler strategey $\Lambda_{\mathcal{E}'}$ in SM+ that also achieves a positive non-termination probability.*

Based on these lemmas, we are finally ready to prove Gorilla's liveness guarantee.

▶ **Theorem 22.** *The Gorilla protocol terminates with probability 1.*

Proof. By contradiction, assume that there exist a GM environment and a Byzantine strategy Θ in Gorilla that achieve a positive non-termination probability. By Lemma 20, there exist a GM+ environment and a strategy Ψ for the Byzantine nodes in GM+ that achieve a positive non-termination probability. Similarly, by Lemma 21, there exists an SM+ environment and a scheduler strategy Λ in SM+ that achieve a positive non-termination probability. But this is a contradiction, since Sandglass terminates with probability 1 in SM+ (Theorem 9). Thus, Byzantine strategy Θ cannot force a positive non-termination probability; Gorilla terminates with probability 1. ◀

6 Conclusion

Gorilla Sandglass is the first Byzantine-tolerant consensus protocol to guarantee, in the same synchronous model adopted by Nakamoto, deterministic agreement and termination with probability 1 in a permissionless setting. To this end, Gorilla leverages VDFs to extend the

approach of Sandglass, the first protocol to provide similar safety guarantees in the presence of benign failures. Neither Gorilla nor Sandglass are practical protocols, however: they exchange a very large number of messages and the number of rounds they require to decide is large even under favorable circumstances, and can, in general, be exponential. Is there a *practical* permissionless protocol that can achieve deterministic safety and tolerate fewer than a half Byzantine nodes?

References

1. James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.
2. James Aspnes, Gauri Shah, and Jatin Shah. Wait-free consensus with infinite arrivals. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 524–533, 2002.
3. Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.
4. Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.
5. Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. *Foundations of Probabilistic Programming*. Cambridge University Press, 2020.
6. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.
7. Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 23–41. Springer, 2019.
8. Soubhik Deb, Sreeram Kannan, and David Tse. Posat: proof-of-work availability and unpredictability, without the work. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 104–128. Springer, 2021.
9. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
10. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings CRYPTO '92: 12th International Cryptology Conference*, pages 139–147. Springer, 1992.
11. Matthias Fitzi, Peter Ga, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *Cryptology ePrint Archive*, 2018.
12. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.
13. Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.
14. Benjamin Lucien Kaminski. *Advanced weakest precondition calculi for probabilistic programs*. PhD thesis, RWTH Aachen University, 2019.
15. Rami Khalil and Naranker Dulay. Short paper: Posh proof of staked hardware consensus. *Cryptology ePrint Archive*, 2020.

16 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I*, pages 357–388. Springer, 2017.

17 Andrew Lewis-Pye and Tim Roughgarden. Resource pools and the cap theorem. *arXiv preprint arXiv:2006.10698*, 2020.

18 Andrew Lewis-Pye and Tim Roughgarden. Byzantine generals in the permissionless setting, 2021. doi:10.48550/ARXIV.2101.07095.

19 Jieyi Long. Nakamoto consensus with verifiable delay puzzle. *arXiv preprint arXiv:1908.06394*, 2019.

20 Dahlia Malkhi, Atsuki Momose, and Ling Ren. Byzantine consensus under fully fluctuating participation. *Cryptology ePrint Archive*, 2022.

21 Michael Mirkin, Lulu Zhou, Ittay Eyal, and Fan Zhang. Sprints: Intermittent blockchain pow mining. *Cryptology ePrint Archive*, 2023.

22 Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2295–2308, 2022.

23 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

24 Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.

25 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology – EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 643–673. Springer Verlag, 2017.

26 Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 380–409. Springer, 2017.

27 Youer Pu, Lorenzo Alvisi, and Ittay Eyal. Safe Permissionless Consensus. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2022.33.

28 Youer Pu, Lorenzo Alvisi, and Ittay Eyal. Safe permissionless consensus. Cryptology ePrint Archive, Paper 2022/796, 2022. URL: https://eprint.iacr.org/2022/796.

29 Youer Pu, Ali Farahbakhsh, Lorenzo Alvisi, and Ittay Eyal. Gorilla: Safe permissionless byzantine consensus, 2023. arXiv:2308.04080.

30 Ronghua Xu and Yu Chen. Fairledger: a fair proof-of-sequential-work based lightweight distributed ledger for iot networks. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 348–355. IEEE, 2022.