SAFE PERMISSIONLESS CONSENSUS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Youer Pu

August 2024

SAFE PERMISSIONLESS CONSENSUS

Youer Pu, Ph.D.

Cornell University 2024

Nakamoto's consensus protocol, known for operating in a permissionless model where nodes can join and leave without notice. However, it guarantees agreement only probabilistically. Is this weaker guarantee a necessary concession to the severe demands of supporting a permissionless model? This thesis shows that it is not with the Sandglass and Gorilla Sandglass protocols.

Sandglass emerges as the first permissionless consensus algorithm that transcends Nakamoto's probabilistic limitations by guaranteeing deterministic agreement and termination with probability 1, under general omission failures. It operates under a *hybrid synchronous* communication model, where, despite the unknown number and dynamic participation of nodes, a majority are consistently correct and synchronously connected.

Further building on the framework of Sandglass, Gorilla Sandglass is the first Byzantine fault-tolerant consensus protocol that preserves deterministic agreement and termination with probability 1 within the same synchronous model adopted by Nakamoto. Gorilla addresses the limitations of Sandglass, which only tolerates benign failures, by extending its robustness to include Byzantine failures. 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.

BIOGRAPHICAL SKETCH

Youer Pu is a PhD candidate in Computer Science at Cornell University. She earned her Bachelor's degree in Computer Science from Shanghai Jiao Tong University in 2014. Afterward, she began her PhD studies at the University of Texas at Austin, where she spent two years before transferring to Cornell to continue her research.

To the days I thought the proofs were wrong and the nights I thought the proofs were correct.

ACKNOWLEDGEMENTS

Pursuing a PhD has been an incredible journey filled with challenges. Without the companionship and support of those around me, nothing would have been possible or meaningful.

First and foremost, I extend my deepest appreciation to my advisor, Lorenzo Alvisi. My PhD journey would never have begun if Lorenzo had not visited Shanghai Jiao Tong University in the summer of 2012. It has been an honor to work with such a dedicated researcher who consistently strives to do the right thing. Lorenzo's kindness, support, and boundless energy have lifted my spirits countless times throughout this journey.

I am also profoundly grateful to my collaborator and committee member, Ittay Eyal. During the most challenging months, our weekly meetings have been the ballast that I hold on to. Thank you for being both a battle companion and a true friend.

I extend my heartfelt thanks to my collaborators, Natacha Crooks and Ali Farahbakhsh. With Natacha, I joyfully began my PhD journey, and with Ali, I happily concluded it. Thank you both for all the intellectual joy. Doing research with you has been both a fortune and an honor. You have filled our research experience with precious, golden moments that I will cherish for a very long time.

I sincerely appreciate all my friends who offered earnest support during this journey. Special thanks to Peihan Miao, Yue Guo, Yun Liu, Qiuhan Ding, Xiaoting Qi, and Lilin Lin. Your late-night messages and unwavering support helped me persevere. You are my strongest fortress, shielding me from any hardship and giving me the strength to face another battle. I am also grateful to Yige Hu, Yuming Sheng, Yuting Yang, Yawen Fang, Tianze Shi, and Yunhe Liu. This

journey is much more joyful with your companionship.

Finally, I want to thank my parents for their patience and support through all my highs and lows. I also thank my entire family for always believing in me. If earning a PhD is an honor, I hope I have made you proud.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The publication of Bitcoin's white paper [46], besides jump-starting an industry whose market capitalization, according to Forbes [18] was valued at $2.5T in June 2024, presented the distributed computing community with a fundamental question [24]: how should the agreement protocol at the core of Nakamoto's blockchain construction (henceforth, *Nakamoto's Consensus* or *NC*) be understood in light of the combination of consensus and state machine replication [33, 52] that the community has studied for over 30 years? The similarities are striking: in both cases, the goal is to create an append-only distributed ledger that everyone agrees upon, which NC calls a *blockchain*.

But so are the differences. Unlike traditional consensus algorithms, where the set of participants n is known and can only be changed by running an explicit reconfiguration protocol, Nakamoto's consensus is *permissionless*: it does not enforce access control and allows the number and identity of participants to change without notice. It only assumes that the computing power of the entire system is bounded, which effectively translates to assuming the existence of an upper bound $\mathcal{N}$ on the number of participants; and that, at all times. the majority of the computing power be controlled by correct participants.

To operate under these much weaker assumptions, NC adopts a new mechanism for reaching agreement: since the precise value of n is unknown, NC forsakes explicit majority voting and relies instead on a *Proof of Work* (PoW) lottery mechanism [16, 26], designed to drive agreement towards the blockchain whose construction required the majority of the computational power of all participants. With PoW, a process can work for a short while and probabilistically

succeed in solving a puzzle. Finally, whereas traditional consensus protocols guarantee agreement deterministically, NC can do so only probabilistically; furthermore, that probability approaches 1 only as termination time approaches infinity. Is settling for these weaker guarantees the inevitable price of running consensus in a permissionless setting?

Some attempts have been made to settle this question. Lewis-Pye and Roughgarden showed that deterministic and permissionless consensus cannot be achieved in a synchronous network in the presence of Byzantine failures [37]. Nonetheless, previous work (§5) has achieved deterministic safety and termination with probability 1 under different models. Pass et al. propose the Sleepy Model [48], where participants join and leave ("sleep"); the model assumes a public key infrastructure (PKI), and guarantees of the consensus protocol presented for this model are as probabilistic as those of pure proof of work. Momose et al. [43] guarantee termination only if the set of processes stabilizes. Malkhi et al. [41] and Losa et al. [19], while leveraging either authenticated channels or digital signatures, propose a solution with constant expected latency.

However, since the attempts above either limit Byzantine behavior, restrict how nodes join and leave, or rely on authentication, it is still unclear whether it is possible to achieve deterministic safety and ensure termination with probability 1 in a fully permissionless setting without these constraints.

We answer this question in the affirmative by first introducing Sandglass, a permissionless consensus algorithm that ensures *deterministic* agreement and achieves termination with probability 1 within a hybrid-synchronous benign model. We then build on Sandglass by introducing Gorilla Sandglass, which extends the same guarantees to a general synchronous Byzantine model.

1.1 Contributions

In summary, this thesis makes the following contributions:

- **It formalizes Nakamoto's permissionless protocol in the vocabulary of traditional consensus.** Nakamoto's blockchain protocol is structured as a chain of blocks, where each block contains a hash of the content of the preceding block, thus, in effect, pointing to its immediate predecessor. This design choice, introduced to protect the integrity of the data stored in the blockchain, has significant implications on how NC can be understood as a consensus protocol.

  Intuitively, NC can be seen as running multiple instances of consensus, with the blockchain playing the role of the consensus ledger. This intuition, however, misses a key difference. When running traditional (repeated) consensus to fill a ledger, each of the consensus decisions needed to fill the ledger is independent of the others; not so in NC, where each new ledger entry must point back to all the entries that precede it.

  Therefore, a more nuanced interpretation of NC as a consensus protocol is needed. As detailed in Chapter 2, we argue that adding a new block b in NC involves more than just starting a new instance of consensus to fill the next entry of the ledger; it also, implicitly, results in proposing (again), the blocks in the chain that b belongs to for the instances of consensus that determine the content of the earlier entries of the ledger.

  The process of proposing also differs from traditional consensus protocols, where it typically involves explicit message exchanges. In Nakamoto's blockchain protocol, participants communicate only sporadically, remain-

ing largely passive and silent unless they possess a valid proof of work. However, as we explain in Chapter 2, even though participants do not explicitly send proposal values, the mining work they quietly perform effectively amounts to proposing in all ongoing consensus instances.

Indeed, although NC is not typically understood as a round-based protocol, we argue that the addition of a new block to the chain can be seen not just as the beginning of a new consensus instance, but as the start of a new round for all the consensus instances responsible for deciding the previous blocks in the chain. Thus, the "longest-chain wins" mechanism is more than just a simple rule for extending the chain; it plays a crucial role in signaling the progress of consensus.

- **It exposes the connection between PoW and a voting mechanism that can be implemented via message passing.**

  Once we frame Nakamoto's blockchain protocol through the lens of consensus, we can begin to unravel the role that PoW plays in facilitating consensus.

  In NC, receiving a longer chain signals that a participant has successfully found a valid PoW. On the other hand, we have argued above that accepting a longer chain in NC should be interpreted, in traditional consensus terms, as entering a new round of consensus.

  Typically, for a consensus participant to advance to a new round, some condition must hold. In a synchronous system, that condition may be the passage of a certain amount of time (the round's length); in an asynchronous system, it may be the receipt of messages from a quorum. In our context, then, a natural question arises: when looking at NC from the perspective of a traditional, round-based consensus protocol, what is the

equivalent of receiving a valid PoW to enable a participant to move from one round to the next?

This dissertation answers this question by introducing a novel approach that expresses the work a participant performs to solve a cryptographic puzzle in terms of messages sent by the participant; as a result, it becomes possible to express the expected total amount of work needed for a group of participants to solve a cryptographic puzzle (*i.e.,* the PoW's *difficulty*) as a threshold of messages that need to be generated and received.

Although the idea of requiring the receipt of a threshold of messages (typically, one message from a majority of participants) to regulate passage from one round to the next is common in asynchronous consensus protocols, the approach we propose departs from it in significant ways. Specifically, since we aim to develop a permissionless consensus protocol, in which the number of participants is unknown and can change at all times, it becomes impossible to condition progress on the receipt of a message from a majority of participants: there is no way to know what that majority, at any time, is!

Instead, our approach requires each participant, while executing in a round, to *continuously* send and receive messages. Progress to the next round occurs only after receiving a threshold of messages that is independent of the number of participants currently in the system, but only depends on the upper bound N on the number of participants.

This novel approach suggests a new way to leverage the assumption that a majority of computing power is in the hands of correct participants: those who do not receive messages from a majority will inevitably take more time to reach the threshold than those who do, and thus start losing

ground on them. And, unlike PoW, who can only give probabilistic guarantees that the majority of correct nodes will be first to generate a block (move to a new round), this mechanism does not allow participants that have fallen behind to recover lost ground through a sequence of strokes of luck.

- **It introduces Sandglass, the first protocol that achieves deterministic agreement in a permissionless setting under hybrid synchrony.** Sandglass is a novel protocol that achieves deterministic agreement in a permissionless setting. It draws inspiration from both Nakamoto's Proof of Work (PoW) mechanism and Ben-Or's classic consensus protocol [5]. Sandglass operates within a hybrid synchronous network model and assumes a benign failure model.

  While its round-based structure resembles that of Ben-Or, Sandglass uses the novel threshold mechanism discussed above to determine when a participant can advance to a new round, eliminating the need to know the number of participants.

  In keeping with Ben-Or's approach to ensuring safety, participants in Sandglass give priority, when proposing, to the value they have seen being unanimously proposed in the previous round. In Ben-Or, where progress to the next round is only possible after receiving a message from a majority, a participant can decide as soon as it has seen the same unanimous value being proposed for two consecutive rounds (*i.e.,* the value reaches priority 2). In Sandglass, where faulty nodes can still make progress without receiving any messages from the correct majority, the priority of a value must be much higher before it is safe for a process to decide on that value. As we will see in Chapter 3, this stringent condition is necessary to

ensure that even if faulty nodes temporarily keep pace with correct nodes, they cannot overturn the decisions of the correct nodes.

- **It introduces Gorilla Sandglass, the first protocol to achieve deterministic safety and liveness with probability 1 in a permissionless *Byzantine* model.** Gorilla Sandglass builds on the approach of Sandglass. Sandglass already defends against attacks such as ignoring messages and strategically choosing when to send messages to correct nodes. When moving to the full Byzantine model, the major challenge left becomes controlling the rate at which Byzantine nodes can send messages. To address this, Gorilla Sandglass uses Verifiable Delay Functions (VDFs). These functions ensure that all nodes can only send a message after a verifiable delay has elapsed, thereby limiting the speed of generating messages.
- **It introduces novel proof strategies.**

  In Sandglass The Sandglass proof diverges substantially from the typical proof style of traditional consensus protocols. In these protocols, the number of participants is known: this knowledge makes it possible to enable mechanisms that prevent isolated faulty nodes, those who only communicate with other faulty nodes (*i.e.*, choose not to receive messages from correct nodes, who are a majority) from making progress. When the number of participants is unknown and can constantly fluctuate, however, these mechanisms become unfeasible: any fixed majority threshold may prove either too low to block the progress of faulty participants (whenever the fluctuating number of participants grows to twice the threshold or more) or too high to allow the progress of correct participants (whenever the number of correct participants is lower than the threshold).

Instead of outright preventing the progress of isolated faulty nodes, Sandglass' novel mechanism guarantees that isolated faulty nodes advance more slowly from round to round than correct nodes – eventually, they lag so far behind that they can no longer influence the values proposed by correct nodes, thereby guaranteeing safety. Additionally, the impossibility of relying on quorum intersection adds complexity to the proof of termination. It becomes necessary to demonstrate that faulty nodes, which may stubbornly propose conflicting values, do not impede the system's liveness.

In Gorilla Sandglass We prove the correctness of Gorilla Sandglass by reducing it to Sandglass: we show that any violation of safety and liveness in Gorilla Sandglass would correspond to one in Sandglass. Having proved that Sandglass is safe and terminates with probability 1, we then conclude that the same must hold for Gorilla.

To this effect, since both Sandglass and Gorilla executions proceed over a sequence of steps, we would ideally be able to align at step boundaries each Gorilla execution with a corresponding Sandglass execution, in a way that ensures that any violation of safety or liveness in Gorilla would also occur in Sandglass.

However, the mapping is far from straightforward. For example, it becomes necessary to address the possibility that Byzantine nodes may act across step boundaries, interleaving VDF computations instead of producing one VDF (and hence one message) at a time. A further novel aspect of the proof comes up when proving termination – in particular, the challenge consists in proving that if a Gorilla execution, once it has been aligned to an execution in Sandglass, ter-

minates with probability 1, so does the pre-alignment Gorilla execution.

1.2 Overview

This thesis is structured as follows: Chapter 2 provides foundational knowledge on Nakamoto's blockchain protocol alongside Ben-Or's probabilistic termination consensus protocol. Chapter 3 presents Sandglass, a novel algorithm that ensures deterministic safety and termination with probability 1 within a permissionless context. Chapter 4 introduces Gorilla Sandglass which incorporates a Verifiable Delay Function (VDF) cryptographic primitive to extend the safety and liveness guarantees of Sandglass to a synchronous Byzantine model. Chapter 5 discusses related work, and Chapter 6 concludes the thesis. The full proof of Sandglass and Gorilla can be found in Appendix A and Appendix B, respectively.

CHAPTER 2

BACKGROUND

2.1 Consensus protocols

A consensus protocol addresses the challenge of ensuring that a set of nodes, each starting with an initial value and communicating by sending messages to one another, arrive at a unanimous, irreversible decision on the same value despite potential failures.

Formally, consensus is specified in terms of three properties:

Validity: If all initial values are v and a correct node decides, it decides v.

Agreement: If a correct node decides on v, no correct node decides differently.

Termination: Every correct node eventually decides on a value.

Whether consensus can be achieved, and, if so, how and at what cost, depends on timing and failure models [7, 34, 38, 40, 54].

Timing assumptions Consensus protocols can be designed to operate in either *asynchronous* or *synchronous* systems.

In asynchronous systems (*i*) there is no upper bound on how long it takes for a message to travel from a correct sender to a correct receiver; (*ii*) there is no bound on the relative processing speed of the nodes in the system; and (*iii*) the local clocks of these nodes are not synchronized. Any system that is not asynchronous is synchronous: in these systems, it is easy to structure protocols

as a sequence or rounds, each of length Δ, where Δ is an upper bound of the time needed for a message to travel between two correct nodes. A system does not live in one of these two camps forever. In particular, *partially synchronous* systems [15] operate asynchronously until some unknown *Global Stabilization Time* (GST), whenceforth they are assumed to behave synchronously.

Failure assumptions A node is considered *faulty* if it deviates from its specification; otherwise, it is *correct*. There are two fundamental ways in which such deviations (or *faults*) can manifest.

***Omission faults*:** A node fails to take actions it is supposed to take. These faults, which are often referred to as *benign*, include *crashes* as well as less clear-cut situations, such as when a node selectively fails to send or receive some message.

Commission faults: A node takes actions that it is not supposed to take. These faults, which are often referred as *malicious*, include, for example, situations when a node *equivocates* when performing a broadcast, sending distinct messages to different nodes.

The *Byzantine* failure model [15] allows faulty nodes to experience omission as well as arbitrary commission failures.

In a synchronous system, consensus can be achieved even in the presence of Byzantine failures, as long as f, the number of faulty nodes is less than a third of n, the total number of nodes in the system.

In an asynchronous system, to the contrary, the celebrated FLP impossibility result [17] establishes a sobering truth: no deterministic protocol can provide a

solution to consensus, even if only a single node fails, and only by crashing.

Traditional attempts to sidestep FLP have focused either on strengthening the model (by equipping each node with sufficiently powerful, if unreliable, failure detectors [8]) or on weakening the problem (for example, by only guaranteeing Termination after GST [35], when the system is no longer asynchronous).

Ben-Or's protocol [5] introduces a form of weakening particularly relevant to the rest of this dissertation. By giving nodes the possibility of flipping a coin, instead of insisting on a purely deterministic solution, this purely asynchronous protocol guarantees safety (captured by Validity and Agreement) while promising Termination with probability 1.

2.2 Nakamoto's blockchain protocol

Nakamoto's blockchain protocol [46] introduces new and interesting dimensions to the consensus problem. The goal of the protocol is to create the abstraction of a shared ledger. The ledger is simply an *append-only* log for storing data: new items can only be added to the end of the ledger and no previous item can be modified. The data recorded in the ledger consists of transactions–these are typically financial transactions, but, more generally, they are operations that cause atomic changes to the state whose evolution the ledger tracks. For performance reasons, the ledger does not grow at the granularity of individual transactions; rather, an entry of the ledger records a group of transactions batched in a *block*, which (hence the name *blockchain* used to characterize the resulting ledger).

The need for consensus arises from the need to guarantee agreement on which block is associated with which entry of the ledger.

Model Traditional consensus protocols operate in a controlled environment: the total number of nodes engaging in consensus is known, and, although up to f nodes can fail, they cannot otherwise leave or join the protocol at will: participation is regulated by some form of trusted authority. Consensus is typically achieved by having participating nodes vote on their (current) preferred values, which different protocols then aggregate in their own unique way to produce a single consensus value.

Nakamoto's protocol departs drastically from this traditional setting. It aims to support *fully permissionless* participation. No authority controls who is allowed to read from, verify the contents of, or append a new entry to the ledger. Thus, the number of participating nodes is unknown, and nodes can join and leave the protocol at will.[1]

There *are*, however, assumptions that Nakamoto's protocol does depend on for correctness. First, the network must be synchronous [21, 31, 47]. Second, there must exist a known upper bound on the amount of computational resources that can, at any time, be used towards running the protocol [47]. Note that, under the assumption that all participating nodes have identical computational power, this is equivalent to assuming a known upper bound on the maximum number of nodes that can, at any point, be participating in the protocol. Without this assumption, an unbounded increase in computational power

[1] Nakamoto's requirement that a majority of the system's computational power be controlled by correct nodes does, indirectly, pose some limit on a node's freedom to change its membership status.

could lead to multiple participants solving the puzzle simultaneously, resulting in divergent chains that prevent the system from achieving consensus. We remark that assuming the existence of this upper bound says nothing about the specific amount of computational power (or, similarly, about the number of participants) used in running the protocol at any specific time, except that it is no larger than the upper bound. Third, correct nodes should, at all times, control a majority of the computational power used in running the protocol [2]. Finally, all nodes have access to a random oracle which, for any given input, produces a unique value taken from a uniformly random distribution.

Protocol Nakamoto's protocol addresses two main concerns in constructing a blockchain.

The first is to ensure the blockchain's integrity: once a block has been added to the blockchain, it should not be possible to alter its content without the tampering being detected. Thus, every block added to the ledger after the very first block (referred to as the *genesis block*), stores a cryptographic hash of the previous block, which serves both as a backward pointer and as a guarantee of that block's integrity.

The second concern is more immediately tied to consensus. In a fully permissionless setting, nothing prevents a single party from participating in the consensus protocol using arbitrarily many identities [50]; thus, traditional consensus schemes that assign each participant equal voting rights can be easily subverted by a party bent on obtaining a specific consensus outcome.

[2] Recent studies suggest that correct nodes should actually enjoy a slightly wider margin than the smallest of majorities for correctness to hold.

Nakamoto solves this problem by choosing as the consensus value the one proposed by whoever is able to solve a *Proof-of-Work* (POW) puzzle (described below).

The nodes that compete to solve the puzzle are called *miners*. Each of them keeps a local version of the blockchain and attempts to extend it with a new block NB that contains (*i*) a set of transactions and (*ii*) a pointer to the last block in its local version of the blockchain.

To succeed, the miner needs to solve a puzzle. Specifically, for other correct nodes to consider NB as successfully added to the blockchain, the miner needs to find a value (the presumptive solution) such that, once NB (which includes a set of transactions and a backward pointer to a previous block) *and* the presumptive solution are given together as input to the random oracle, the value produced in output is lower than a given threshold.

The only known method to find a solution to this puzzle is through brute force, *i.e.* by trying random values. Once a miner succeeds, they broadcast their extended chain. If a node receives a longer chain than its current one, it adopts this new chain — this mechanism is referred to as the *longest-chain-wins* rule.

A block is deemed decided once it is followed in the blockchain by a *confirmation threshold* of T additional blocks. In Bitcoin, the cryptocurrency that relies on Nakamoto's blockchain protocol, the threshold consists of five blocks. At this threshold, the likelihood that Byzantine nodes, even if they constitute up to 10% of the computing power, will be able to alter the chain to exclude a confirmed block reduces to below 0.4% [23].

Correctness The protocol probabilistically ensures agreement on an ever-growing prefix of the chain. The chance of Byzantine nodes outpacing correct nodes to "rewrite history" by producing a longer chain depends on their chance to be faster at finding PoW solutions; since there is no faster way than brute force to find a solution, that chance is proportional to the computing power they control. Here is where Nakamoto's assumption that correct nodes control a majority of computational power plays a crucial role: because of it, correct nodes are more likely to find PoW solutions faster than Byzantine nodes bent on rewriting history. As more blocks are added behind a block in a correct node's chain, the likelihood that Byzantine nodes can produce a longer alternative chain diminishes, solidifying the block's permanence in the correct chains.

However, this level of assurance still falls short of a deterministic guarantee: there always exists a probability, albeit small, that attackers might succeed in overriding a given blockchain by producing a longer alternative chain. Furthermore, for any protocol that terminates after a finite number of steps, that probability is not 0.

2.3 Nakamoto's protocol: a traditional Consensus perspective

In this section, we aim to interpret the entire process of a Nakamoto Consensus (NC) execution through the framework of traditional consensus mechanisms.

Multiple consensus instances on a chain Nakamoto's ledger abstraction is a familiar one: it is at the core of state machine replication [33, 52], the most general approach to build reliable distributed services.

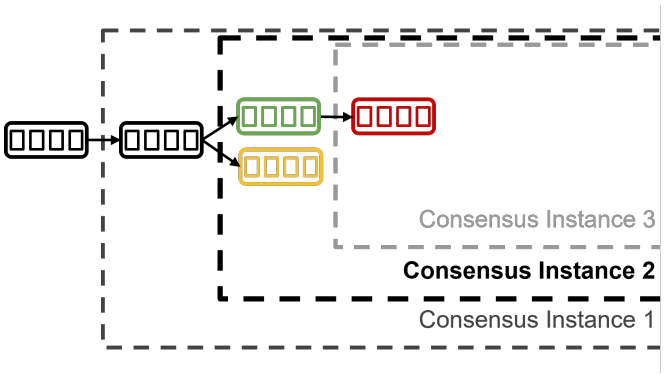


Figure 2.1: Consensus instances in Nakamoto

In *that* context, the goal is to determine the single total order in which correct, deterministic replicas are to process clients' requests. Each entry in the ledger is decided by running an *independent* instance of consensus, where instance i determines the content of the i-th entry. Multiple instances can execute concurrently, and the vagaries of network latency can cause later ledger entries to be decided before earlier ones—this is not a problem, as long as correct replicas execute the request held in the ledger's i-th entry only after having executed those held in the $i - 1$ entries that precede it.

Nakamoto's ledger adds a twist to this picture: because in Nakamoto, every block in the chain points to its unique predecessor, the instances of consensus that decide the content of the different entries in the ledger are no longer independent. A miner that, in consensus instance i, proposes a particular block b to fill the i-th entry in the chain is implicitly participating in all preceding instances of consensus: it is proposing to fill entry $i - 1$ with the block that b points back to—and so on, recursively, until the very first entry of the ledger.

For example, in Fig 2.1, the node that generated the red block is effectively proposing this block for Consensus Instance 3. By extending a particular chain, it also proposes the green block for Consensus Instance 2 and similarly for the

blocks preceding it.

Nakamoto Consensus as a round-based protocol It is useful to describe how an instance of Nakamoto consensus converges on a decision as a round-based process. Consider a block chain of length $l - 1$. All participants working on the computational puzzle that must be solved to extend the chain with a new block can be seen as engaged in the first round of the $l-th$ instance of consensus. Once a correct participant p succeeds in extending the chain, it broadcasts a message with the new chain to everyone, still within the first round of instance l. By receiving and accepting this new chain, all correct participants (i) move to the second round of instance l and (ii) enter the first round of consensus instance $l + 1$.

In general, a participant working on extending a chain of length $(l - 1)$ is in the $(l + 1 - i)$-th round of the i-th consensus instance, for each $1 \leq i \leq l$ and the messages it sends are to be interpreted, within each consensus instance $1 \leq i \leq l$, as having been sent in the $(l + 1 - i)$-th round of that instance of consensus.

This round-based perspective offers a natural interpretation of Nakamoto's longest-chain-wins rule.

Consider a Byzantine participant that intentionally ignores a longer chain of length l broadcast by some correct participant and attempts instead to extend a shorter chain of length-$(l - 1)$. When the Byzantine node eventually succeeds in generating a chain of length l and broadcasts it, this chain will be ignored by the correct nodes under the longest-chain wins rule, because it is not longer than the chains that correct nodes currently maintain.

From a round-based perspective, this behavior is easy to justify. Correct participants see the Byzantine-generated message as a first-round message of the l-th consensus instance (and more generally, as a $(l+1-i)$-th round message of the i-th consensus instance for each $1 \leq i \leq l$). All correct participants, however, have already advanced to the second round of the l-th consensus instance (and similarly, to the $(l + 2 - i)$-th round of the i-th consensus instance, for each $1 \leq i \leq l$).

Therefore, the longest-chain wins rule can be seen as acting essentially as a mechanism for disregarding messages from past rounds in a traditional round-based protocol.

Note that participants may skip certain rounds within a consensus instance. For example, if a correct node that is extending a chain of length $(l - 1)$ receives a chain of length $(l + 1)$, it progresses to the third round of the l-th consensus instance and simultaneously enters the first round of $(l + 2)$-th consensus instance. This occurs without the node ever being in the second round of the l-th consensus instance.

Lottery voting In Nakamoto Consensus, the process for selecting proposal values resembles a lottery voting mechanism. A key property of the puzzle to be solved in order to extend a chain c of length l by some block b, is that any unit of computing power has a constant probability of finding a solution. Thus, each unit of computational power spent on solving that puzzle can be seen as a lottery ballot with $l + 1$ values printed on it: the first l values read as the corresponding block in the existing chain; the last value reads b.

Thus, the probability that a block b^* will be selected as the proposal value

for the *next* round of a consensus instance i is proportional to the computing power expended on the chain that either already holds b^* in its entry i, or is in the process of being extended by choosing a value for entry i—that is, it is proportional to the number of ballots that read b^* as their proposed value for the appropriate consensus instance.

Note that even participating nodes extending different chains might have the same block b in their ballot for the i-th consensus instance if their chains share a prefix up to the i-th block.

We observe that NC's approach to selecting proposal values for the next round—randomly choosing from all ballots— is quite different from the alternative approach of first conducting a leader election and then adhering to the leader's proposals. The key distinction emerges when two puzzle solutions are discovered almost simultaneously; in such cases, NC considers both proposal values equally valid, whereas leader election mechanisms would treat the leader's decision as distinctly superior at all times.[3] This characteristic aligns NC more closely with consensus protocols like Ben-Or [5], which also allows participants, in certain circumstances, to randomly select their proposal values, and sets it apart from leader-based consensus protocols like Paxos [34], which rely on a single leader's decisions.

Decision Nakamoto Consensus uses a confirmation threshold T to determine when a decision has been reached. Specifically, a block b at height k on the longest chain is considered decided if there are at least T blocks following b in the chain. From the perspective of the i-th consensus instance, all values

[3]The distinction cannot be fudged by treating the puzzle solving exercise as a leader election protocol, since such protocol must produce a *single* leader.

proposed from round $(T + 1)$ on are its decision values.

Therefore, from the perspective of the i-th consensus instance, there are multiple ways of violating agreement:

Multiple Decisions by a Single Node : A correct node p proposes block b in the j-th round of the i-th consensus instance, where $j > T$, and then proposes a different block $b' \neq b$ in the $(j + 1)$-th round of the same consensus instance. This scenario leads p to decide multiple times on different values within the same consensus instance. This situation may occur when a node, maintaining a chain of length at least $i + T$, receives a longer chain where the i-th block differs from the one in its current chain.

Disagreement among Different Nodes : Agreement is also violated when a correct node p proposes block b in the j-th round of the i-th consensus instance, where $j > T$, and another correct node q proposes a different block $b' \neq b$ in the same round of the same consensus instance. This results in p and q deciding on different values within the same consensus instance. This situation may occur when two nodes have chains whose suffixes diverge by more than the last T blocks.

2.4 Ben-Or's classic consensus protocol

The Ben-Or binary consensus protocol (Protocol 1) circumvents the FLP impossibility result by weakening the Termination clause of the specification of Consensus: the guarantee that every correct node will eventually decide is only given with probability 1.

Protocol 1 Ben-Or Consensus Protocol [5]

```
1:  v_i ← input_i; r ← 0;
2:  loop
3:      broadcast (r, phase = 1, v_i)                          ▷ Phase 1
4:      wait until ⌊n/2⌋ + 1 messages from Phase 1 are received
5:      if all the messages from Phase 1 have the same proposal v then   ▷ Phase 2
6:          broadcast (r, phase = 2, v)
7:      else
8:          broadcast (r, phase = 2, ⊥)
9:      wait until ⌊n/2⌋ + 1 messages from Phase 2 are received
10:     if ∃(r, 2, v ≠ ⊥) then
11:         v_i = v
12:     else
13:         v_i = randomly pick one from V
14:     if v ≠ ⊥ for all the messages from Phase 2 then
15:         decide(v)
16:     r = r + 1
```

Model Ben-Or's protocol solves binary consensus in a permissioned setting within an asynchronous network—that is, without loss of generality [45], participants are limited to proposing either 0 or 1; the number n and identities of participants are known; and there is no upper bound on message delivery times. Ben-Or assumes that majority of the n participants are correct and that the f participants who are faulty, where $f < n/2$, can suffer benign failures, *i.e.*, general omission failures, including crashes.

Protocol The protocol progresses in asynchronous rounds. Each round is divided into two phases; in each phase, every non-faulty node broadcasts a message and collects the messages sent by its peers in that phase. Because the network is asynchronous and fewer than half of the nodes can fail, nodes do not wait for more than a majority of responses.

In Phase 1 of each round (Line 3–4), each node broadcasts its current proposal

value and waits for the values broadcast by a majority of nodes (including itself). If all these values are identical (*i.e.*, if a majority is proposing the same value), the node adopts that value (call it v) as its proposal for Phase 2 of the round; otherwise, it adopts $\perp$.

In Phase 2, each node broadcasts the proposal value adopted at the end of Phase 1 (Line 5-8) and once again waits for the values broadcast by a majority of nodes. There are three cases.

1. If a node receives only proposals consisting of the same value $v \neq \perp$, (meaning that a majority of nodes report that they have each observed a majority of nodes proposing v), then the node decides v (Line 14-15) and retains v as its proposal value for Phase 1 of the next round.
2. If a node receives some $\perp$ and some v (meaning that at least some node observed a majority of nodes proposing v in Phase 1) then the node adopts v as its proposal value for Phase 1 of the next round (Line 11).
3. If a node receives only $\perp$ (meaning that a majority of nodes received conflicting proposal in Phase 1) then the node's proposal value for Phase 1 of the next round is selected at random (Line 13).

Correctness Ben-Or's protocol guarantees Validity and Agreement, while ensuring Termination with probability 1.

Validity If all nodes initially propose the same value v, each node can only receive Phase 1 messages containing v. Consequently, in Phase 2, all nodes that send a message (excluding those that crash, experience a send omission, or are stuck in Phase 1 because of receive omission) will broadcast Phase 2 messages

with v. By the end of Phase 2, all correct nodes will have collected a majority of Phase 2 messages containing v, and will thus all decide v, satisfying Validity.

Agreement Suppose r is the earliest round that some correct node p decides, and p decides v in round r. We need to show that all correct nodes will eventually decide v.

First, it is easy to see that no other correct node can decide something other than v in round r. To decide v, p must have received a majority of messages in Phase 2 of round r proposing v (Line 15); thus, since all majorities intersect, in a benign failure model there cannot exist in that phase a majority proposing some $v' \neq v$. In fact, applying again the observation that all majorities intersect, but this time on proposals received in Phase 1 of round r, it is impossible for two distinct values v and v', both other than $\perp$, to be proposed in Phase 2 of any round.

Further, again because all majorities intersect, any node that in Phase 2 of round r receives a majority of messages (in particular, every correct node) must receive at least one of the messages proposing v that p received. Therefore, all such nodes will adopt v as their proposal value for Phase 1 of the next round (*i.e.*, round $(r + 1)$) (Line 13).

Therefore, all nodes capable of moving to round $(r + 1)$ will propose value v, and will only receive Phase 1 messages proposing v. Then, all nodes that manage to send messages will broadcast Phase 2 messages proposing v. By the end of Phase 2, all correct nodes will collect a majority of Phase 2 messages proposing v, and thus decide v by the end of round $(r + 1)$, satisfying Agreement.

Termination We saw in our proof for Validity how, if all the nodes start with the same proposal v in Phase 1 of round r, all correct nodes decide v (and thus satisfy Termination) in that round. Therefore, Ben-Or's protocol will only fail to terminate if not all nodes propose the same value for an infinite number of rounds.

Consider the probability that nodes will have the same proposal value in round $(r + 1)$ under different round r scenarios:

No non-$\perp$ messages received in Phase 2 All nodes randomly select a value to propose for round $r + 1$ (Line 13). The probability that they will select the same value is non-zero.

Some non-$\perp$ messages received in Phase 2 All nodes that receive a Phase 2 message with v will propose v in round $r + 1$. We showed in the Agreement proof that it is impossible to receive a Phase 2 message with a value different from v, other than $\perp$. The probability that all nodes without a non-$\perp$ Phase 2 message will randomly select v is also non-zero. Thus, the probability that all nodes will select the same value is non-zero.

Therefore, regardless of what happens in round r, the probability that the protocol will terminate in round $(r+1)$ is non-zero, meaning the probability that the protocol will *never* terminate is 0. Thus, Ben-Or's protocol terminates with probability 1.

2.5 Reframing Ben-Or's consensus protocol in terms of *priority*

If we take a step back and examine the selection process of proposal and decision values in Ben-Or's protocol, we observe that values endorsed by a majority are selected as proposal values, and a value that receives majority support in both phases of a round is chosen as the decision value. This progression, where a value repeatedly endorsed by a majority is prioritized over others, forms the core of how decisions are reached.

Sandglass and Gorilla (Chapter 3, 4) adopt this same principle of prioritizing values that are continuously endorsed by a majority across multiple rounds. To better understand and highlight this process, we introduce an explicit *priority* variable and rewrite Ben-Or's protocol to incorporate it. This variable counts the number of consecutive phases in which a value is unanimously endorsed by a majority. Intuitively, in Ben-Or's protocol, when the priority of a value v is 1, v is adopted as the proposal value for the next phase. If the priority of v reaches 2, v is decided.

Making a value's priority explicit reveals the remarkable similarity between the two phases that constitute each round of Ben-Or's protocol: in both phases, the value with the higher priority is the one chosen to be proposed or decided. Drawing out this similarity allows for a more streamlined presentation of the protocol as a sequence of identical, elemental rounds no longer decomposed in distinct phases. This reframed version of Ben-Or (see Algorithm 2) not only is more concise, but also serves as an excellent foundation for the Sandglass protocol, where, as we will see in Chapter 3, the explicit handling of priority becomes pivotal in adapting consensus mechanisms to a permissionless setup.

Protocol 2 Reframed Ben-Or Consensus Protocol with priority

```
1:  v_i ← input_i; r ← 0; priority_i ← 0
2:  loop
3:      broadcast (r, v_i, priority_i)
4:      wait until a set of ⌊n/2⌋ + 1 messages, M_i, is received
5:      Consider the set of values, L, that have the largest priority.
6:      Randomly pick any value v from L
7:      v_i = v
8:      if v = v_i for all the messages in M_i then
9:          priority_i = min_{(r,v_i,priority)∈M_i}(priority) + 1
10:     else
11:         priority_i = 0
12:     if priority_i = 2 then
13:         decide(v_i)
14:     r = r + 1
15:     if r mod 2 == 0 then
16:         priority_i = 0
```

Starting from round 0, even-numbered rounds correspond to the first phases of the original protocol, and odd-numbered rounds correspond to the second phases.

We now show that this new formulation is equivalent to the original one.

In the new formulation, the structure of each round is simple.

1. Nodes broadcast their proposal value with its associated priority.
2. Nodes wait to receive a majority of proposals.
3. Nodes select the value to be proposed in the next round by choosing randomly among the received proposals with highest priority.
4. Nodes associate a priority with the selected value on the basis of criteria, which we will discuss in a moment, chosen so that even and odd rounds

operate, respectively, as Phase 1 and Phase 2 of a round of the original Ben-Or.

5. If the priority associated to the selected value is 2, then that value is decided.

Let's see in greater detail how the protocol's behavior differs in even and odd rounds. Their first three steps are identical; the differences arise in the priorities that can be associated to the values proposed in those rounds. In reading the discussion below, recall that the value that is being *proposed* in an odd (respectively, even) rounds, depends on the values (with associated priorities) *received* in the previous even (respectively, odd) round.

Odd rounds Since all values proposed in even rounds have the same priority (*i.e.*, 0) (see below the discussion of **Even rounds**), the set L that determines what is going to be proposed in the following odd round contains *all* the values a node received in the preceding even round. There are two possibilities:

1. If all the received values are the same (say, v), then the node sets its proposal value to v (since it is picking at random among identical values!) and increases v's priority to 1 (Line 9). This is equivalent to adopting v as the proposal value at the beginning of Phase 2 of the original Ben-Or after receiving a unanimous majority of proposals for v in Phase 1.
2. Otherwise, it chooses at random one of the received values and assigns to it priority 0. This is equivalent to adopting $\perp$ as the proposal value at the beginning of Phase 2 of the original Ben-Or after receiving different proposal values in Phase 1.

Even rounds As mentioned above, all values proposed in even rounds have priority 0. This is true in the very first round (Line 1) and is enforced by setting the priority to 0 at the beginning of all even rounds (Line 16). What remains is determining the actual value that a node proposes: this depends on the values the node received in the preceding odd round and on their associated priority.[4] The protocol requires the proposal value to be chosen randomly among the collected proposals with highest priority (Lines 5-6). What can those priorities be? There are three possibilities.

1. All values received in the preceding odd round have priority 0; this case corresponds, in the original Ben-Or, to receiving only Phase 2 messages reading $\perp$. In this case, as in the original Ben-Or formulation, the value proposed in the next round is chosen at random and could be either of the two initial values of the binary consensus being executed.

2. Some (but not all) values received in the preceding odd round have priority 1; this case corresponds, in the original Ben-Or (Lines 10-11), to receiving some non-$\perp$ Phase 2 messages. Note that, since the priority of a proposal value is increased only if that is the only value received from a majority of nodes, the by-now-familiar majority intersection argument ensures that there cannot be multiple distinct values with priority 1 received in an odd round.

3. All messages received in the preceding odd round propose the same value v with priority 1; this case corresponds, in the original Ben-Or, to receiving a majority of matching Phase 2 messages proposing the same value v. This is the condition under which the original Ben-

[4] The value a node proposes in the very first round is its initial proposal.

Or decides v. The same happens in our reformulation: v's priority is increased from 1 to 2 (Line 9), which then causes v to be decided (Line 12-13).

In conclusion, this reformulation of Ben-Or protocol, is equivalent to the original. At the same time, it makes explicit what the original leaves implicit—namely, that a value proposed by a majority should be prioritized in both the proposal and decision-making processes.

Further, by making explicit the process of increasing the priority of a value, it makes it possible to think of protocols in which priority values can grow larger than 2. This newly-found flexibility will prove particularly valuable in the design of the Sandglass protocol, which will be explored thoroughly in Chapter 3.

2.6 Verifiable Delay Function

Verifiable Delay Functions (VDFs) are cryptographic primitives that require a predetermined number of sequential operations to compute on some input, while producing a result that can be efficiently and quickly verified by anyone [6, 49, 55]. This characteristic of having a high computation cost but a low verification effort makes VDFs particularly beneficial in permissionless settings, as they allow any verifier to quickly verify whether a specific amount of time has been devoted to a particular input.

In the Gorilla protocol, VDFs are employed to prevent Byzantine nodes from conducting Sybil attacks. In such attacks, a single Byzantine node may generate numerous pseudonymous identities to send a vast number of messages, aiming

to disproportionately sway the voting process. To counteract this threat, the protocol mandates that each message sent must be accompanied by a VDF result, which is evaluated based on the message content. This safeguard ensures that, even in a permissionless environment, Byzantine nodes are unable to exploit the system by multiplying their influence through Sybil attacks. Thus, the integrity of the voting process is maintained, protecting the consensus mechanism from manipulation.

By demonstrating that a specific amount of computational effort has been expended on given message content, VDFs thus fulfill a role similar to Proof of Work (PoW). The primary distinction between VDFs and PoW lies in the nature of their computations: VDFs require sequential processing that cannot be parallelized, unlike PoW, which can be accelerated with a larger amount of computational resources.

Moreover, for the Gorilla protocol, a significant advantage of Verifiable Delay Functions (VDFs) is their deterministic nature. VDFs are evaluated through a fixed number of operations, ensuring consistent and predictable computational effort across all nodes. This deterministic process is crucial as it translates the majority assumption based on participant numbers effectively into a majority assumption based on message counts. Therefore, if a majority of nodes are honest and adhere to the protocol, this majority is mirrored in the number of valid messages produced, effectively preventing any minority (possibly malicious) group from exerting undue influence.

In contrast, Proof of Work (PoW) mechanisms depend on a probabilistic approach, where the necessary number of operations can vary significantly. This inconsistency can allow faulty or malicious nodes to progress faster than honest

nodes, potentially influencing the protocol disproportionately. Thus, the predictable, fixed effort required for VDFs provides a more secure and fair mechanism for protocols like Gorilla, ensuring that correct nodes cannot be outpaced by faulty ones simply through variance in computational effort.

Here is a brief summary of how VDF works. A VDF is defined by a triple of algorithms: *Setup*, *Eval*, and *Verify*:

Setup$(\lambda, t) \rightarrow pp = (ek, vk)$ generates a pair of public parameters $pp = (ek, vk)$, where ek refers to the evaluation key and vk refers to the verification key. These parameters are generated based on a security parameter λ and a delay parameter t.

The security parameter λ, determines the cryptographic strength of the VDF, ensuring that the computation cannot be accelerated through parallel processing or advanced cryptanalytic techniques. It ensures that the probability of successfully bypassing the delay function is negligible in λ, *i.e.*, decreasing faster than the inverse of any polynomial in λ.

The delay parameter t, on the other hand, specifies the minimum amount of time required to compute the VDF, measured by the number of sequential steps or operations. This parameter enforces a fixed time delay, ensuring that the function takes a predictable amount of time to compute, regardless of the computational resources available.

Together, the delay and security parameters balance the computational effort required, resistance to parallelism, and cryptographic strength, ensuring that the VDF meets both performance and security requirements.

Eval$(ek, x) \rightarrow (y, \pi)$ takes an input x and an evaluation key ek from public parameters pp, computes the output y after t sequential steps.

Alongside the output value y the function may also generate a proof π. This proof is used to verify that the output y was correctly computed following the prescribed sequential steps. In some cases, π might be empty if the VDF does not require additional proof or if the proof is implicitly included in the output value y.

Verify$(vk, x, y, \pi) \rightarrow \{Yes, No\}$ is a deterministic algorithm that takes a verification key vk, an input x, an output y, and a proof π as its parameters. Using these inputs, the function checks whether the proof π correctly verifies that the output y was obtained from the input x through the prescribed sequential steps. If the proof is valid, Verify returns "Yes", indicating that the VDF computation was performed correctly and the output y is trustworthy. If the proof is invalid or missing in a scheme where proof is required, the function returns "No", signaling that the output y may not be correct or the computation might have been tampered with.

In VDF schemes where the proof is optional or implicitly included in the output y, the verification function may use alternative methods to validate the output directly using the verification key vk and the input x. In such cases, Verify can still return "Yes" or "No" based on these alternative verification methods.

This verification process is designed to be significantly faster than the Eval.

This thesis leverages several key properties of Verifiable Delay Functions (VDFs), the formal definitions of which are detailed in Boneh et al. (2018) [6]:

Correnctness The Verify algorithm should return "Yes" for any output produced by the Eval function.

Soundness For any x, the probability of finding a y where $Verify(pp, x, y, \pi) = Yes$ but $Eval(pp, x) \neq y$ is negligible in λ.

Sequentiality Correct nodes can compute $(y, \pi) \leftarrow \text{Eval}(ek, x)$ in t sequential steps whereas no adversary with a parallel machine and a polynomial number of processors can differentiate the output y from random in significantly fewer steps.

Efficient Verification Verification is fast compared to the evaluation, requiring only poly-logarithmic time relative to t [49] or even constant time [9, 11, 55].

CHAPTER 3

SANDGLASS

In this chapter, we present *Sandglass*, a permissionless consensus algorithm that guarantees *deterministic* agreement and terminates with probability 1. Sandglass operates in a model based on Nakamoto's. The model allows an arbitrary number of participants to join and leave the system at any time and stipulates that at no time the number of participants exceeds an upper bound N (though the *actual* number n of participants at any given time is unknown). Further, like Nakamoto's, it is *hybrid synchronous*, in that, at all times, a majority of participants are correct and able to communicate synchronously with one another. We call these participants *good*; our protocol's safety and liveness guarantees apply to them. Participants that are not good (whether because they crash, perform omission failures, and/or experience asynchronous network connections) we call *defective*. Sandglass proceeds in asynchronous rounds, with a structure surprisingly reminiscent of Ben-Or's classic consensus protocol [5]. We briefly review it here, and the full discussion is in §2.4.

In Ben-Or's protocol, nodes propose a value by broadcasting it; in the first round, each node proposes its initial value; in subsequent rounds, nodes propose a value chosen among those received in the previous round. Values come with an associated priority, initialized to 0. The priority of v depends on the number of consecutive rounds during which v was the only value received by the node proposing v – whenever a node receives a value other than v, it resets v's priority back to 0. When proposing a value in a given round, node p selects the highest priority value received in the previous round; if multiple values have the same priority, then it selects randomly among them. A node can

safely decide a value v after sufficiently many consecutive rounds in which the proposals it receives unanimously endorse v (i.e., when v's priority is sufficiently high); and termination follows from the non-zero probability that the necessary sequence of unanimous, consecutive rounds will actually eventually occur.

Of course, embedding this structure in a permissionless setting introduces unprecedented challenges. Consider, for example, how nodes decide. In Ben-Or, a node decides v after observing two consecutive, unanimous endorsements of v; it can do so safely because any two majority sets of its fixed set of n nodes intersect in at least one correct node. This approach is clearly no longer feasible in a permissionless setting, where n is unknown and the set of nodes can change at any time.

Instead, Sandglass's approach to establishing safety is inspired by one of the key properties of Nakamoto's PoW: whatever the value of n, whatever the identity of the nodes participating in the protocol at any time, the synchronously connected majority of *good nodes* will, in expectation, be faster than the remaining nodes in adding a new block to the blockchain.

Think now of adding a block b at position i of the blockchain as implicitly starting a new round of consensus for all the chain's positions that precede i; for each position, the new round proposes the corresponding block in the hash chain that ends at b. In this light, the greater speed in adding blocks that PoW promises to the majority of connected nodes translates into these nodes moving faster from one asynchronous round to the next in each of the consensus instances.

This insight suggests an alternative avenue for achieving deterministic con-

sensus among good nodes – without relying on quorum intersection. Node p should decide on a value v only after it has seen v unanimously endorsed for sufficiently many rounds that, if p is good, the lead p (and all other good nodes) have gained over any defective node q proposing some other value is so large that q's proposals can no longer affect the proposal of good nodes.

Why can't the same approach be used to achieve deterministic consensus in Nakamoto's original protocol? Because Nakamoto's PoW mechanism, notwithstanding its name, is an indirect and imperfect vehicle for proving work. As evidence of performed work, Nakamoto presents the solution to a puzzle: this solution, however, could just have been produced as a result of a lucky guess. Thus, however unlikely, it is always possible in NC for defective nodes proposing a value other than v to catch up with, or even overtake, good nodes and reverse their decisions.

To avoid this danger, Sandglass relies on a different PoW mechanism, which ties the ability to propose a value to a *deterministic* amount of work. In particular, Sandglass nodes can propose a value in any round other than the first only after they have received a specific threshold of messages from the previous round. Therefore, each proposed value implicitly represents all the work required to generate the messages needed to clear the threshold. The threshold value is chosen as a function of the upper bound $\mathcal{N}$ on the number of nodes that at any time run the protocol, in such a way that, whatever is their actual number n, any node that does not receive messages from good nodes will inevitably take longer than them in moving from round to round.

The full power of this PoW mechanism, however, comes from pairing it with the idea, which we borrow from Ben-Or, of associating a priority with the val-

ues being proposed. With a fixed set of n nodes, Ben-Or leverages priorities and quorum intersection to safely decide a value v once it has reached priority 2, because it can guarantee that henceforth every node executing in the same round as a correct node will propose v. In a permissionless setting, we show that the combination of priorities and our PoW mechanism allows Sandglass to offer good nodes the same guarantee (though, as we will see, v will be required to reach a significantly higher priority value!). Intuitively, by the time v reaches the priority necessary to decide, any node q that manages not to fall behind (and thus become irrelevant) to the unanimous majority of good nodes who have kept proposing v must have received *some* of the messages proposing v from some good nodes. Furthermore, to keep up, q must have received such messages often enough that, given how the priority of received values determines what a node can propose, it would be impossible for q to propose any value other than v.

In summary, this chapter makes the following contributions: (*i*) it formalizes Nakamoto's permissionless model in the vocabulary of traditional consensus analysis; (*ii*) it introduces novel proof strategies suitable for this new model; (*iii*) it exposes the connection between PoW and a voting mechanism that can be implemented by message passing; and (*iv*) it introduces Sandglass, the first protocol that achieves deterministic agreement in a permissionless setting under hybrid synchrony.

In this chapter, we present the model of Sandglass in Section 3.1, the protocol in Section 3.2, and the correctness of Sandglass in Section 3.3.

3.1 Model

The system comprises an infinite set of nodes $p_1, p_2, \ldots$. Time progresses in discrete steps; in each step, a subset of the nodes is *active* and the rest are *inactive*. At each step, active nodes are partitioned into *good* and *defective* subsets.

We assue a hybrid synchronous model. *Good* nodes are correct, and the network that connects them to one another is synchronous; at all times, a majority of active nodes are good. *Defective* nodes may suffer from benign failures, such as crashes and omission failures, or simply lack a synchronous connection with some good node.

The system progress is orchestrated by a *scheduler*. In each step, the scheduler can activate any inactive node p_i (we say that p_i has *joined* the system) and deactivate an active node (which then *leaves* the system). The scheduler chooses which nodes to activate and deactivate arbitrarily, subject only to the following three constraints: (*i*) The upper bound of active nodes in any step is $\mathcal{N}$; (*ii*) there is at least one active node in every step; and (*iii*) in every step the majority of active nodes is good.

In each step where it is active, each node p_i executes the stateful protocol shown as procedure *Step* in Sandglass's pseudocode (see Algorithm 3). It can execute computations, update its state variables, and communicate with other nodes with a broadcast network. In particular, since Sandglass assumes benign failures, every active node, whether good or defective, waits for a full step to elapse before sending its next message.

The network allows each active node to *broadcast* and *receive* unauthenticated

messages. Node p_i broadcasts a message m with a $Broadcast_i(m)$ instruction and receives messages broadcast by itself and others with a $Receive_i$ instruction. The network does not generate or duplicate messages, *i.e.*, if in step t a node p_i receives message m with $Receive_i$, then m was sent in some step $t' < t$.

The communication model is designed to capture the design of Nakamoto's consensus, which relies on an underlying network layer to propagate and store blocks. Nakamoto's network layer provides a shared storage of data structures, called blocks, and guarantees delivery of published blocks within a bounded time. Each block includes cryptographically secure references to all blocks seen by its creator. This allows a newly joined node to receive and validate the entire history of published blocks. Thus, in our model, the scheduler determines when each message is delivered to each node under the following constraints.

First, propagation time is bounded between any pair of good nodes. Formally: if a good node p_i calls $Broadcast_i(m)$ in step t, and if a good node p_j calls $Receive_j$ in step $t' > t$, then m is returned, unless it was already received by p_j in an earlier call to $Receive_j$. Thus, a newly activated good node is guaranteed, upon executing its first *Receive*, to receive all messages from other good nodes broadcast in the steps prior to its activation.

Second, the network is reliable, but there is no delivery bound unless both nodes are good. Formally: For any two nodes p_i and p_j, where at least one of p_i and p_j is defective, and for a message m broadcast by p_i, if node p_j calls $Receive_j$ infinitely many times, then m is eventually delivered.

Each node is initiated when joining the system with an initial value $v_i \in \{a, b\}$. An active node p_i can decide by calling a $Decide_i(v)$ instruction for some value v.

The goal of the nodes is to reach a consensus based on these values:

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

Definition 2 (Validity)**.** *If all nodes that ever join the system have initial value v and any node (whether good or defective) decides, then it decides v.*

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

3.2 Protocol

To form an intuition for the mechanics of Sandglass, it is useful to compare and contrast it with Ben-Or. From a distance, the high-level structure of the two protocols is strikingly similar: execution proceeds in asynchronous rounds; progress to the next round depends on collecting a threshold of messages sent in the current round; safety and liveness depend on the correctness of a majority of nodes; and nodes decide a value v when, for sufficiently many consecutive rounds, all the messages they collect propose v. But looking a little closer, the differences are equally striking. On the one hand, Sandglass's notion of node correctness and its hybrid synchronous model are stronger than Ben-Or's. Sandglass assumes a majority of good nodes that are not only free from crashes and omissions, but also synchronously connected to one another. On the other hand, in Sandglass, unlike Ben-Or, the number n of nodes running the protocol is not only unknown, but may be changing all the time. These differences motivate four key aspects that separate the two protocols:

Choosing a threshold In Ben-Or, a node advances to a new round only after having received a message from a majority of nodes. This strict condition

for achieving progress is critical to how Ben-Or establishes Agreement. Any node that, from a majority of the nodes in round r, receives a set of messages that unanimously propose v, can be certain that (*i*) there cannot exist in r also a unanimous majority proposing a value other than v and (*ii*) no node can proceed to round $(r + 1)$ unaware that v is among the values proposed in round r. Nodes that isolate themselves from a majority simply do not make any progress; and since all majority sets intersect, nodes cannot make contradictory decisions.

Unfortunately, this approach is unworkable in Sandglass: when the cardinality and membership of the majority set can change at any time, receiving messages from a majority can no longer serve as a binary switch to trigger progress. More generally, thresholds based on the cardinality of the set of nodes from which one receives messages become meaningless. Instead, Sandglass allows nodes to broadcast multiple messages during a round, one in each of the round's steps, and lets nodes move to round $(r + 1)$ once they have collected a specified threshold of messages sent in round r.

Think of the threshold $\mathcal{T}$ of messages that allows a node to move to a new round as the number of grains of sands in a sandglass: a node (figuratively) flips the sandglass at the beginning of a round, and cannot move to the next until all $\mathcal{T}$ sand grains have moved to the bottom bulb. The value of $\mathcal{T}$ is the same for all nodes; the speed at which messages are collected, however—the width of their sandglass's neck—is not, and can change from step to step: if all nodes broadcast messages at the same rate, the larger the number of nodes that one receives messages from in a timely fashion, the faster it will be to reach the threshold. Thus, while in Sand-

glass setting a threshold cannot altogether prevent nodes that don't receive messages from a majority from making progress, it ensures that they will progressively fall behind those who do.

Exchanging messages In each step of the protocol, a node currently in round r (*i*) determines, on the basis of the messages received so far, what is the largest round $r_{max} \geq r$ for which it has received the required threshold of messages and (*ii*) broadcasts a message for round r_{max}, which includes the node's current proposed value, as well as the critical *metadata* discussed below.

Keeping history Unlike Ben-Or, Sandglass allows nodes to join the system at any time. To bring a newly activated node up to speed, each message broadcast by a node p in round r carries a *message coffer* that includes (*i*) the set of messages (at least $\mathcal{T}$ of them) p collected in round $r-1$ to advance to round r; (*ii*) recursively, the set of messages in those messages' coffers; and (*iii*) the set of messages p collected so far for round r.

Respecting priority In Ben-Or, a node decides v if, for two consecutive rounds, v is the only value it collects from a majority set. To ensure the safety of that decision, Ben-Or assigns a *priority* to the value v that a node p proposes: if v was unanimously proposed by all the messages p collected in the previous round, it is given priority 1; otherwise, 0. Nodes that collect more than one value in round r, propose for round $r + 1$ the one among them with the highest priority, choosing by a coin flip in the event of a tie. Sandglass uses a similar idea, although its different threshold condition requires a much longer streak of consecutive rounds where v is unanimously proposed before v's priority can be increased. To keep track of the length of that streak, every message sent in a given round r carries a *unanimity*

counter, which the sender computes upon entering r.

3.2.1 Selecting the Threshold

Unlike Ben-Or, Sandglass's threshold condition can not altogether prevent nodes from making progress. It is perhaps surprising that, by leveraging only the assumption that at all times a majority of nodes are good (i.e., correct and synchronously connected with each other) without ever knowing precisely how many they actually are, Sandglass retains enough of the disambiguating power of intersecting majorities to ultimately yield deterministic agreement.

In essence, Sandglass succeeds by causing defective nodes that isolate themselves from the majority of nodes in the systen to fall eventually so far behind that they no longer share the same round with good nodes. At the same time, it ensures that, once some good node has decided on a value v, nodes that manage to keep pace with good nodes will never propose anything other than v.

Of course, to obtain this outcome it is critical to set $\mathcal{T}$ appropriately. Consider two nodes, one good and one defective, and suppose they flip their sandglass at the same time—*i.e.*, they enter a new round in the same step. We want that, independent of how the number of active nodes may henceforth vary at each step, if the defective node only receives messages from other defective nodes (*i.e.*, if it fails to hear from a majority of nodes), it will reach the threshold $\mathcal{T}$ at least one step later than the good node will. The following lemma shows that setting $\mathcal{T}$ to $\lceil \frac{\mathcal{N}^2}{2} \rceil$ (where $\mathcal{N}$ is the upper bound on the maximum number of nodes active in any step) does the trick.

Lemma 1. *For any k, consider any time interval comprising $(k+1)$ consecutive steps.*

Let the number of messages generated by good nodes and defective nodes in each step of the interval be, respectively, $g_0, g_1, ..., g_k$ and $d_0, d_1, ..., d_k$. Setting the threshold $\mathcal{T}$ to $\lceil \frac{\mathcal{N}^2}{2} \rceil$ ensures that, if $\Sigma_{i=1}^{i=k-1} g_i < \mathcal{T}$, then $\Sigma_{i=0}^{i=k} d_i < \mathcal{T}$.

Proof. Note how the lemma does not count the messages generated by good nodes in the steps at the two ends of the interval. Recall that moving from the current round to the next requires a node to receive at least a threashold $\mathcal{T}$ of messages sent in the current round. Note that good nodes that in step 0 enter a new round r are unable to count against the threshold for round r messages generated by good node that in step 0 are still in round $r-1$; thus, we drop good messages from step 0. Similarly, we drop step k because good nodes may only need one of the messages sent by good nodes in step k to move to a new round – and have no use for the remaining messages in g_k.

We begin by observing that, when k is either 0 or 1, the lemma trivially holds, since in all steps defective nodes generate fewer than $\mathcal{N}$ messages. For example, when $k = 1$, $d_0 + d_1 < \frac{\mathcal{N}}{2} + \frac{\mathcal{N}}{2} = \mathcal{N} \leq \lceil \frac{\mathcal{N}^2}{2} \rceil$. We then prove the lemma for $k \geq 2$.

Let $\bar{g} = \frac{\Sigma_{i=1}^{i=k-1} g_i}{k-1}$ and $\bar{d} = \frac{\Sigma_{i=1}^{i=k-1} d_i}{k-1}$ denote, respectively, the average number of messages generated by good nodes and by defective nodes during the $k-1$ steps that include all but the interval's first and last step. Expressed in terms of $\bar{g}$ and $\bar{d}$, the lemma requires us to show that, if $\bar{g} \cdot (k-1) < \mathcal{T}$, then $\Sigma_{i=0}^{i=k} d_i = d_0 + \bar{d} \cdot (k-1) + d_k < \mathcal{T}$ when $\mathcal{T}$ is chosen to equal $\lceil \frac{\mathcal{N}^2}{2} \rceil$.

Assume $\bar{g} \cdot (k-1) < \mathcal{T}$; then $k - 1 < \frac{\mathcal{T}}{\bar{g}}$. Substituting for $(k-1)$ in the formula

that computes the messages generated by defective nodes, we have:

$$\begin{aligned}
\Sigma_{i=0}^{i=k} d_i &= \bar{d} \cdot (k-1) + d_0 + d_k \\
&< \bar{d} \cdot \frac{\mathcal{T}}{\bar{g}} + d_0 + d_k \quad (\text{since } (k-1) < \frac{\mathcal{T}}{\bar{g}}) \\
&\leq \bar{d} \cdot \frac{\mathcal{T}}{\bar{g}} + \frac{\mathcal{N}-1}{2} + \frac{\mathcal{N}-1}{2} \quad (\text{since defective nodes are always a minority}) \\
&\leq \bar{d} \cdot \frac{\mathcal{T}}{\bar{g}} + \frac{\mathcal{T}}{\frac{\mathcal{N}^2}{2}}(\mathcal{N}-1) \quad (\text{since } \mathcal{T} = \lceil \frac{\mathcal{N}^2}{2} \rceil \geq \frac{\mathcal{N}^2}{2}) \\
&= \mathcal{T}(\frac{\bar{d}}{\bar{g}} + \frac{2(\mathcal{N}-1)}{\mathcal{N}^2}).
\end{aligned}$$

Then, to establish that $\Sigma_{i=0}^{i=k} d_i < \mathcal{T}$, it suffices to prove that $\frac{\bar{d}}{\bar{g}} + \frac{2(\mathcal{N}-1)}{\mathcal{N}^2} < 1$.

Since for any i, $d_i \leq g_i - 1$ and $d_i + g_i \leq \mathcal{N}$, we know that $\bar{d} \leq \bar{g} - 1$ and $\bar{d} + \bar{g} \leq \mathcal{N}$. Dividing both inequalities by $\bar{g}$ yields $\frac{\bar{d}}{\bar{g}} \leq \min(1 - \frac{1}{\bar{g}}, \frac{\mathcal{N}}{\bar{g}} - 1)$. Note that the largest value of $\min(1 - \frac{1}{\bar{g}}, \frac{\mathcal{N}}{\bar{g}} - 1)$ occurs when $1 - \frac{1}{\bar{g}} = \frac{\mathcal{N}}{\bar{g}} - 1$; solving for $\bar{g}$ and plugging the solution back in, gives us: $\min(1 - \frac{1}{\bar{g}}, \frac{\mathcal{N}}{\bar{g}} - 1) \leq \frac{\mathcal{N}-1}{\mathcal{N}+1}$.

Therefore, we have that $\frac{\bar{d}}{\bar{g}} + \frac{2(\mathcal{N}-1)}{\mathcal{N}^2} \leq \frac{\mathcal{N}-1}{\mathcal{N}+1} + \frac{2(\mathcal{N}-1)}{\mathcal{N}^2} = \frac{\mathcal{N}^3+\mathcal{N}^2-2}{\mathcal{N}^3+\mathcal{N}^2} < 1$. □

3.2.2 Protocol Mechanics

Protocol 3, besides showing how Sandglass initializes its key variables, presents the code that node p_i executes to take a step. Every step begins with adding all received messages, as well as the messages in their message coffers, to a single set, Rec_i (lines 4 - 5). Going over the elements of that set, p_i determines the largest round r_{max} for which it has received at least a threshold $\mathcal{T}$ of messages, and, if the condition at line 6 holds, sets the current round to $(r_{max} + 1)$ (line 7). Upon entering a new round, p_i does four things. First, after resetting its

Protocol 3 Sandglass: Code for node p_i

1: **procedure** INIT($input_i$)
2: $v_i \leftarrow input_i$; $priority_i \leftarrow 0$; $uCounter_i \leftarrow 0$; $r_i = 1$; $M_i = \emptyset$; $Rec_i = \emptyset$; $uid_i = 0$
3: **procedure** STEP
4: **for all** $m = (\cdot, \cdot, \cdot, \cdot, \cdot, M)$ received by p_i **do**
5: $Rec_i \leftarrow Rec_i \cup \{m\} \cup M$
6: **if** $\max_{|Rec_i(r)| \geq \mathcal{T}}(r) \geq r_i$ **then**
7: $r_i = \max_{|Rec_i(r)| \geq \mathcal{T}}(r) + 1$
8: $M_i = \emptyset$
9: **for all** $m = (\cdot, r_i - 1, \cdot, \cdot, \cdot, M) \in Rec_i(r_i - 1)$ **do**
10: $M_i \leftarrow M_i \cup \{m\} \cup M$
11: Let C be the multi-set of messages in $M_i(r_i - 1)$ with the largest priority.
12: **if** all messages in C have the same value v **then**
13: $v_i \leftarrow v$
14: **else**
15: $v_i \leftarrow$ one of$\{a, b\}$, chosen uniformly at random
16: **if** all messages in $M_i(r_i - 1)$ have the same value v_i **then**
17: $uCounter_i \leftarrow 1 + \min\{uCounter | (\cdot, r_i - 1, v_i, \cdot, uCounter, \cdot) \in M_i(r_i - 1)\}$
18: **else**
19: $uCounter_i \leftarrow 0$
20: $priority_i \leftarrow \max(0, \left\lfloor \frac{uCounter_i}{\mathcal{T}} \right\rfloor - 5)$
21: **if** $priority_i \geq 6\mathcal{T} + 4$ **then**
22: Decide$_i(v_i)$
23: $uid_i \leftarrow uid_i + 1$;
24: $M_i \leftarrow M_i \cup Rec_i(r_i)$
25: broadcast $(p_i, uid_i, r_i, v_i, priority_i, uCounter_i, M_i)$

message coffer M, p_i collects in the coffer all the messages it received from the previous round—as well as the messages stored in the coffers of those messages (lines 8 - 10). Second, p_i chooses the value v that it will propose in the current round (lines 11 - 15): it picks the highest-priority value among those collected in its coffer for the previous round ; if more than one value qualifies, it chooses among them uniformly at random. Third, p_i computes the unanimity counter and the priority for all messages that p_i will broadcast during the current round (lines 16 - 20). The counter represents, starting from the previous round and

going backwards, the longest sequence of rounds for which all corresponding messages in p_i's coffer unanimously proposed v. The priority is simply a direct function of the value of the unanimity counter: we maintain it explicitly because it makes it easier to describe how Sandglass works. Finally, if v's priority is high enough, p_i decides v (lines 21- 22). Whether or not it starts a new round, p_i ends every step by broadcasting a message (line 25): before it is sent, the message is made unique (line 23) and p_i adds to the message's coffer all messages received for the current round (line 24).

3.3 Correctness: Overview

Sandglass upholds the definitions of Validity, Agreement, and Termination (with probability 1) given in Section 3.1. We overview the proof below, as its approach differs from proofs of classical, permissioned protocols. We defer the presentation of the full proof to Appendix A, which includes the formal statements of the lemmas we informally state below.

Validity is easily shown by induction on the round number, since if all nodes that join have the same value, there is only one value that can be sent in each round (Lemma 9). Establishing Agreement and Termination is significantly more involved, and hinges on a precise understanding of the kinematics of good and defective nodes—and how that interacts with the ability of good nodes to converge on decision value and on the number of rounds necessary to do so safely. How clustered are good nodes as they move from round to round? At what rate do good nodes gain ground over defective nodes that cut themselves out from receiving messages from good nodes? How often do defective nodes

need to receive messages from good nodes to be in turn able to have their messages still be relevant to good nodes?

The answer to these and similar questions constitute the scaffolding of lemmas and corollaries on which the proofs of Agreement and Termination rely. We discuss it in greater detail below, before moving on to the proofs.

3.3.1 The Scaffolding

The protocol achieves several properties that facilitate the consensus proof.

First, it keeps good nodes close together as they move from round to round. Specifically, in any step two good nodes are at most one round apart (Corollary 2), and if in any step a good node is in round r, then by the next step all good nodes are guaranteed to be at least in round r (Lemma 10). However, note that defective nodes can advance faster than good ones, using a combination of messages from good nodes and messages from defective nodes that do not reach the good nodes. Nonetheless, we show that at any step a defective node is at most one round ahead of any good node (Lemma 12).

Second, the protocol guarantees information sharing among good nodes. This may appear trivial to establish, since good nodes are correct and synchronously connected, but the *laissez-faire* attitude of the permissionless model, with nodes joining and leaving without coordination at any step, complicates matters significantly, making it impossible to prove seemingly basic properties. For example, consider a good node p that, in round r and step T, proposes a value v with a positive *uCounter*. It would feel natural to infer that all good

nodes must have proposed v in the previous round—but it would also be wrong. If p just entered r in step T, it would in fact ignore any value proposed by good nodes that newly joined the systems in step T, but are still in round $r-1$. Fortunately, we show that a much weaker form of information sharing among good nodes is sufficient to carry the day. We say that a node *collects* a message in a round if it receives the message and does not ignore it (messages originated from a lower round number are ignored). We show that, in any round, a good node collects at least one message from a good node (Lemma 13), and that, for any round, there exists a message from a good node that is collected by all good nodes (Corollary 1).

Third, it allows us to establish the basis for a key insight about the kinematics of Sandglass nodes that will be crucial for proving Agreement and Termination: in the long run, the only values proposed by defective nodes that remain relevant to the outcome of consensus are those that have been, in turn, recently influenced by values proposed by good nodes. This insight stands on a series of intermediate results. We already saw (Lemma 1) that, given any sequence of steps, if good nodes cannot generate enough messages to get into the next round, neither can defective nodes, even if they, unlike good nodes, are allowed to count messages generated in the two steps at the opposite ends of the period. It follows that during the steps that good nodes spent in a round, defective nodes can generate fewer than the $\mathcal{T}$ messages necessary to move to the next round (Lemma 17). It all ultimately leads to Lemma 18, which quantifies the slowdown experienced by defective nodes that don't allow themselves to be contaminated by good nodes: it establishes that defective nodes that do not collect any message from good nodes for $k\mathcal{T}$ consecutive rounds fall behind every good node by at least $(k-1)$ rounds.

3.3.2 Agreement

The intuition behind our proof of Agreement is simple. To each value v proposed and collected by Sandglass nodes is associated a *uCounter*, which records the current streak of consecutive rounds for which all the messages collected by the proposer of v were themselves proposing v. Once v's *uCounter* reaches a certain threshold, v's *priority* increases; and once the value v proposed by a node reaches a given priority threshold, then a node decides v (see Algorithm 3, line 21). Since, as we saw, good nodes share information from round to round (recall Corollary 1), proving Agreement hinges on showing that, once a good node decides v, no good node will ever propose a value other than v. To prove that, we must in turn leverage what we learned about the kinematics of Sandglass nodes to identify a priority threshold that makes it safe for good nodes to decide. It should be large enough that, after it is reached, it becomes impossible for a defective node to change the proposal value of any good node.

The technical core of the Agreement proof then consists in establishing the truth of the following (Claim 2):

> Let p_d be the earliest good node to decide, in round r_d at step T_d. Suppose p_d decides v_d. Then, any good node p_g that in any step (whether before, at, or after T_d) finds itself in a round r_g that is at least as large as r_d, proposes v_d for r_g with *priority* at least 1. [1]

It is easy to see that if the above claim holds, then Agreement follows. Say that T_d is the earliest step in which a good node p_d, currently in round r_d, decides

[1] Although proving Agreement does not require that v_d be proposed with priority at least 1, it makes proving the claim easier.

v_d. The claim immediately implies that no good node can decide a value other than v_d in a round greater or equal to r_d, since, from r_d on, every good node proposes v_d. Recall that, since good nodes are never more that one round apart at any step (Corollary 2), the earliest round a good node can find itself at T_d is $(r_d - 1)$; and that, by Lemma 10, every good node is guaranteed to be at least in round r_d by step $(T_d + 1)$. All that is left to show then is that no good node p', which at T_d found itself in round $(r_d - 1)$, can decide some value v' other than v_d. To this end, we leverage the information sharing that we proved exists among good nodes.

By Corollary 1, there is at least a message m generated in round $(r_d - 2)$ by a good node that is collected by all the good nodes. Since p_d at T_d has reached the priority threshold required to decide v_d, m must have proposed v_d; but if so, it would be impossible for good node p', which also must have collected m, to have reached the priority threshold required to decide a different value v'.

Proving Claim 2 is non trivial. The core of the proof consists in showing that any node that proposes a value v' other than the decided value v_d must find itself, at T_d, in a much earlier round than the earliest round occupied by any good node. In fact, we show something stronger: we choose a priority threshold large enough that any node, whether good or defective, that at T_d or later is within earshot of a good node (*i.e.*, whose message m can be collected by a good node), not only proposes v_d, but it does so with a *uCounter* large enough that allows whoever collects m to propose v_d with priority at least 1.

To see why those who propose v' are so far behind good nodes, note that the good node p_d that decided v_d at T_d must have received only messages proposing v_d for a long sequence of rounds, so long as to push v_d's priority over the $(6\mathcal{T}+4)$

threshold required for a decision. Let's zoom in on that sequence of rounds. It took $6\mathcal{T}$ unanimous rounds for v_d to reach priority 1 (see Algorithm 3, line 20); after clearing that first hurdle, v_d's priority increased by 1 every $\mathcal{T}$ rounds.

Consider now the set S of messages collected by p_d during the long climb that took v_d's priority from 1 to $(6\mathcal{T} + 4)$. Any node p' that during this climb proposes something other than v_d faces a dilemma. It can either refuse to collect any message in S — but if it does so, it will advance more slowly than good nodes, and, by the time v_d's priority reaches the decision threshold, it will be so far behind that no good node will collect its messages. Or p' can try to keep up by collecting messages from S — but, if it wants to keep proposing $v' \neq v_d$, it can do so in at most one round during the entire climb: since the first message collected from S would reset v' priority to 0, any further message from S collected by p' in later rounds would have higher priority than the one of v', forcing p' to henceforth propose v_d instead of v'.

In short, since p' can collect messages from S in at most one round, to ensure that any node that in round r_d is within earshot of good nodes will propose v_d it suffices to choose a large enough priority threshold for deciding. In particular, setting the threshold to $(6\mathcal{T} + 4)$ ensures that (*i*) all messages collected by good nodes for round $(r_d - 1)$ will propose v_d, and (*ii*) v_d's *uCounter* in all these messages is at least $(6\mathcal{T} - 1)$, ensuring that all good nodes in round r_d will propose v_d with *uCounter* at least $6\mathcal{T}$, *i.e.*, with priority at least 1.

Finally, a simple induction argument shows that, if all good nodes propose v_d with priority at least 1 from r_d on, then any node that, from step $(T_d + 1)$ on, continues to propose a value other than v_d, will fall ever more behind good nodes, as it will be allowed to collect messages from good nodes only once every

$6\mathcal{T}$ rounds, on pain of being forced to switch its proposed value to v_d.

3.3.3 Termination

The Termination property requires good nodes that stay active to eventually decide. Sandglass's Termination guarantee is probabilistic: for Termination to hold, Sandglass needs to be lucky, so that it can build a sequence of consecutive rounds during which all messages collected by good nodes propose the same value; long enough that the value will reach the priority required for a node to decide. Luck is required because Sandglass allows some randomness in the values that a node proposes: nodes are required to propose the highest priority value from any message collected in the previous round, but, if they receive multiple values with the same priority, they can choose among them uniformly at random.

To help us prove that luck befalls Sandglass with probability 1, we introduce the interdependent notions of *lucky period*, *lucky value*, and *lucky round*. Intuitively, a lucky period is a sequence of steps that leads to a decision: all nodes that are active in the step that immediately follows the end of the lucky period are guaranteed to decide in that step, if not earlier. A lucky round is simply the first round of a lucky period. What is more interesting is the quality that makes a period lucky: during a lucky period, whenever Sandglass allows nodes to use randomness in picking which value they will propose in the current round, they select the same value — the *lucky value* for that round.

A minimum requirement for a round's lucky value is that it should be a *plausible* value on which good nodes may converge, in the sense that it should

not explicitly go counter the value that some good node is required to propose in that round. Concretely, if the messages collected by a good node require it to propose v and all other nodes can randomly choose between v and $\bar{v}$, then the round's lucky value better not be $\bar{v}$. In addition, to encourage the possibility of a lucky period, the lucky value should be *sticky*: we would like random choices to consistently pick the same value, round after round, unless doing so would make the value implausible.

In the end, Sandglass adopts a definition of lucky value (see Appendix A.4) that, in addition to upholding plausibility, has two additional properties that express its stickiness. First, in every round good nodes collect at least one message that proposes the lucky value of the previous round (Observation 2): this guarantees that under no circumstances the previous round's lucky value will simply be forgotten when moving to a new round. The second property, which builds upon the first, establishes that lucky values don't flip easily: (Observation 3): for the lucky value in the current round to change, some good node must have collected a different value with priority at least 1 from the previous round.

To prove that Sandglass guarantees Termination with probability 1, we then proceed in two steps.

First, we show (Observation 5) that the *uCounter* of all good nodes active in the step that immediately follows the end of the lucky period reaches a value that allows these good nodes to decide. To this end, we begin by proving that, in any lucky period, the lucky value after a while becomes *locked*: specifically, we show that the lucky value v_ℓ at round $6\mathcal{T}$ in the lucky period remains the lucky value until the end of the lucky period, and, further, that after that round

all good nodes propose v_ℓ. Then, leveraging techniques similar to those used to prove Agreement, we show that any node p' that proposes a value v' other that v_ℓ must fall behind good nodes during the lucky period. The reason is that, once v_ℓ is locked, p' can collect a message from a good node only every $6\mathcal{T}$ rounds. If it did it more often, p' would collect a message proposing v_ℓ from a good node while v' has priority 0, which would force p' to change its proposal to v_ℓ – even if v' and v_ℓ both had priority 0, and p' could choose randomly among them, it would have to propose v_ℓ in the next round, since v_ℓ is the lucky value. Thus, by choosing a sufficiently long lucky period, we ensure that nodes that propose values other that v_ℓ fall so far behind good nodes that v_ℓ's priority, for any good node that is active in the step right after the end of a lucky period, reaches the threshold necessary for deciding.

Second, we show that lucky periods occur with non-zero probability, since the probability of a certain outcome of random choices for a finite number of nodes during a finite number of steps is non-zero. Since in any infinite execution lucky periods appear infinitely often, it follows that any good node that stays active, no matter when it joins, is guaranteed to eventually decide.

CHAPTER 4

GORILLA SANDGLASS

In this chapter, we present *Gorilla Sandglass* (or simply *Gorilla*) (§4.2), 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 [42] that proves a process waits 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, 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.

Gorilla Sandglass adopts the general approach of Sandglass, 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 (§4.3).

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 works almost without change (§B.1) 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?

4.1 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 [47], 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($\gamma, \boldsymbol{vdf_\gamma^i}$): 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($\boldsymbol{vdf}, \gamma$): 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(γ, vdf_γ^{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 $v_i \in \{a, b\}$ 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 [15]:

Definition 4 (Agreement)**.** *If a correct node decides a value v, then no correct node decides a value other than v.*

Definition 5 (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 6 (Termination)**.** *Every correct node that remains active eventually decides.*

4.2 Gorilla

Gorilla borrows its general structure from Sandglass (see Algorithm 4). 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 nonethe-

less 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 high-

est 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 4 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).

4.2.1 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 4: (*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 4 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: $priority_i \leftarrow \max(0, \left\lfloor \frac{uCounter_i}{\mathcal{T}} \right\rfloor - 5)$
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 [**?**]; Gorilla instead assumes a Byzantine failure model, and a synchronous network model (§4.1). 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.

4.3 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, we could then conclude that neither they can be in Gorilla.

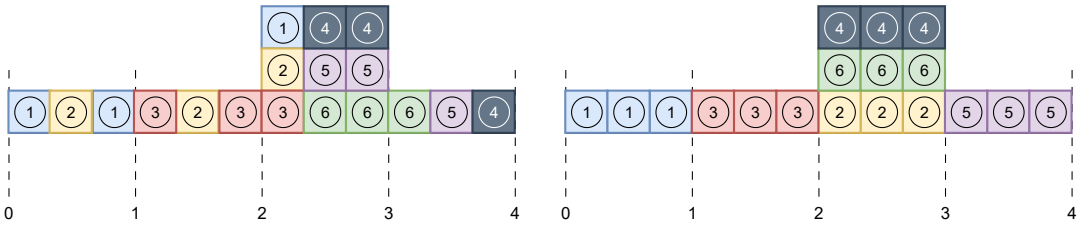
The best laid plans often go awry, and, as we discuss below, ours was no exception—but we were able to nonetheless retain the conceptual simplicity of our initial approach.

4.3.1 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



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

Figure 4.1: An execution that cannot be reorganized in GM (a), and how peeking solves the problem in GM+ (b).

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.

Unfortunately, *well-formedness and equivalence cannot be satisfied by the first mapping*. To see why, consider Figure 4.1a. 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 4.1a 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.

4.3.2 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 4.3.1, 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's *peek* query, Byzantine nodes active in any step s can learn the result of a VDF computed by Byzantine nodes finishing in step s even 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, 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 7. *Consider an execution η_G in GM and an execution η_G^+ in GM+. We say η_G^+ is a reorg of η_G 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 2. *There exists a mapping* REORG *that maps an execution η_G in GM to an execution η_G^+ in GM+, denoted $\eta_G^+ = \text{REORG}(\eta_G)$, such that η_G^+ is a reorg of 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+, a model 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: But 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* (§B.1.2). 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. We map messages by simply translating the data structure:

Definition 8. *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 $\hat{m} =$ 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* MAPM$(\mathcal{M})$.

Thus, we can define the execution mapping:

Definition 9. *Consider an execution η_G^+ in GM+ and an execution η_S^+ in SM+. We say η_S^+ is an interpretation of η_G^+ 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. *Node $\hat{p}$ 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. *Node $\hat{p}$ receives $\hat{m}$ at step s in η_S^+ iff p receives m at step s in η_G^+.*

Lemma 3. *Consider any execution η_G in GM, and an execution η_G^+ in GM+ is a reorg of η_G. There exists a mapping* INTERPRET *that maps η_G^+ to an execution η_S^+ in SM+, denoted as $\eta_S^+ =$ INTERPRET(η_G^+), such that η_S^+ is an interpretation of 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 §B.1.2.

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

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 $\eta_G^+ =$ REORG(η_G) in GM+; then, map η_G^+ again to $\eta_S^+ =$ INTERPRET(η_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 derive a contradiction. This approach is made rigorous in following lemmas, proved in §B.3.

Lemma 4. *Consider an arbitrary Gorilla execution η_G, and $\eta_G^+ =$ REORG(η_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 5. *Consider any execution η_G in GM. If an execution η_S^+ in SM+ is an interpretation of an execution $\eta_G^+ = \text{REORG}(\eta_G)$ in GM+, then the following statements hold:*

1. *If a correct node p decides a value v at step s in η_G^+, then the corresponding $\hat{p}$, 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 2. *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 4, 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 5, $\hat{p}_1^+$ 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 1). Therefore, Gorilla satisfies agreement in GM. □

Theorem 3. *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 7, $\eta_G^+ = \eta_G$ in GM+ is trivially a reorg of η_G. Consider $\eta_S^+ = \text{INTERPRET}(\eta_G^+)$.

By the construction of the INTERPRET mapping (in Lemma 3), 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 9. Therefore, by Validity of Sandglass in SM+ (Theorem 1), no good node decides $v' \neq v$. However, by Lemma 4 and Lemma 5, p decides $v' \neq v$, which leads to a contradiction. Therefore, Gorilla satisfies validity in GM. □

4.3.3 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 [2, 4, 27]. 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 10. *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 11. *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 12. *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 13. *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.4, establishes that Byzantine nodes do not benefit from employing such a randomized strategy.

Lemma 6. *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

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

well-defined and $P_{H_\Theta}(E) = \tilde{P}_{H_\Theta}(\text{COINS}(E))$. 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 14. *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 1).

Lemma 7. *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 2, 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 $\text{REORG}(H_\Theta)$, and consider any execution η in H_Θ. By

Lemma 2, correct nodes in η receive the same messages, at the same steps, as the correct nodes in REORG(η) and, moreover, the coin results in η are exactly the same as the ones in REORG(η). 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 REORG(η). In addition, because REORG(η) 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 4, whenever a correct node decides at some step s in η, its corresponding correct node in REORG(η) 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.4.

Lemma 8. *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 4. *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 7, 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 8, 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 1). Thus, Byzantine strategy Θ cannot force a positive non-termination probability; Gorilla terminates with probability 1. □

CHAPTER 5

RELATED WORK

The consensus problem has been studied for decades, covering both benign and Byzantine faults under different synchrony assumptions. Common across these classic works is the assumption that the set of nodes that participate in running the protocol is either constant or changes through an agreement among the current participants (*permissioned*). In contrast, Sandglass and Gorilla allow for participants to change arbitrarily and without any coordination (*permissionless*) as long as, at all times, a majority of nodes are correct and synchronously connected. More recent papers also explore models where participants can change dynamically at any time, subject to guarantees of a well-behaved majority; unlike Sandglass and Gorilla, those works achieve only probabilistic safety guarantees. We briefly review related prior work in more detail below.

Classic consensus The permissionless nature of our model implies that consensus solutions for classical models (e.g., [28]) do not apply. For synchronous networks, previous solutions rely on the fact that the number of failures is bounded over a period of time. They tolerate up to $(n-1)$ benign failures [54] or Byzantine failures with authentication [14, 36]. For asynchronous networks, Fisher, Lynch, and Paterson [17] show that it is impossible to solve consensus with deterministic safety and liveness, even with a single crash failure. Various protocols (e.g., [32, 51, 53]) thus either solve asynchronous consensus with weaker liveness guarantees than deterministic termination, or provide deterministic termination after a Global Stabilization Time (GST) (e.g., [7]). They use logical rounds, and for each round collect messages from a sufficient number of (authenticated) nodes, tolerating fewer than $\frac{n}{2}$ failures in a benign failure

model [5, 34], and fewer than $\frac{n}{3}$ failures with Byzantine failures and authentication [7, 58]. Although our model is not directly comparable, Sandglass and Gorilla match the $\frac{n}{2}$ bound of a benign model in an asynchronous network, despite assuming synchrony among good nodes.

Impossibility result for permissionless setting Lewis-Pye and Roughgarden [37] show that deterministic consensus cannot be achieved in a permissionless synchronous model with Byzantine nodes, let alone in a partially synchronous model (where communication becomes synchronous only after some GST unknown to the processes). Sandglass and Gorilla show, for the first time, that deterministic safety and termination with probability 1 can be achieved in a permissionless model.

Blockchains A newer line of work, inspired by Nakamoto's introduction of Bitcoin [46], investigates decentralized systems where participants may freely join or leave without notifying existing members.

Bitcoin employs a probabilistic Proof-of-Work (PoW) mechanism, predicated on the idea that a minority group cannot sustain over long duration the pace of the majority in producing proof of work outputs, thereby ensuring system safety with high probability as long as a majority of participants follow the protocol, as confirmed by various studies [13, 21, 31, 47]. Other protocols, like Ethereum and the protocol described in [44], adopt different probabilistic approaches: Ethereum uses Proof-of-Stake (PoS) [56], which secures the network through validators staking their own tokens as a form of security, while the system outlined in [44] utilizes Proof-of-Space-Time (PoST), where participants prove they have allocated disk space over a specific duration to validate trans-

actions. Both mechanisms aim to offer similar probabilistic guarantees tailored to their unique operational frameworks.

Several protocols, inspired by the Proof of Work (PoW) approach, achieve consensus among a large group of principals while requiring the active participation of only a subset of them.

Pass et al. [48] 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. This elegant model requires a public key infrastructure (PKI) and offers probabilistic safety guarantees. Ouroboros [10, 30] approaches blockchain through a proof-of-stake (PoS) mechanism by using internal tokens to randomize participant selection, thus providing probabilistic safety and efficiency. In Algorand [22], committees of users elect one another through successive reconfigurations. Participants are randomly chosen from a large pool, which still allows for a negligible possibility of selecting a committee with a Byzantine majority. Consequently, Algorand too offers only probabilistic safety guarantees.

In contrast, despite their permissionless model, Sandglass and Gorilla guarantee deterministic safety (without relying on PKI) and terminate with probability 1. Sandglass operates under a model that differs from the one adopted by the three above systems. Its network assumptions are weaker, as it allows communication to and from defective nodes to be asynchronous; on the other hand, its failure model is stronger, as it assumes only benign failures. Gorilla, on the other hand, matches the network and failure models adopted by the above

systems, as it assumes a synchronous network and tolerates Byzantine faults.

Participation restriction Few proposals achieve deterministic safety in a permissionless setting [3, 43], but only by limiting the ability of correct nodes to leave the system.

Aspnes et al. [3] explore the consensus problem in an asynchronous benign model where an unbounded number of nodes can join and leave [20], but where at least one node is required to live forever, or until termination. It is easy to see that, without this latter assumption, deterministic safety is impossible in their model.

Momose et al. [43] introduce the notion 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.

In contrast, Sandglass and Gorilla guarantee deterministic safety while allowing *all* nodes to freely join and leave without requiring any correct node to stay in the system or assuming statibility in participation, as long as a majority of active nodes is correct.

Signatures A different line of work by Malkhi et al. [41] and Losa et al. [19] do let nodes join and leave at any time, as in Gorilla. Both studies achieve termination within a constant expected latency. Unlike Gorilla, however, they must rely on a public key infrastructure (PKI) to tolerate fluctuations in the number of adversaries.

Other efforts to make the connection between blockchains and traditional consensus Abraham and Malkhi [1] formalize Nakamoto's Consensus within a classical disturbed systems framework, and in particular abstract the PoW primitive as a Pre-Commit, Non-Equivocation, Leader Election (PCNELE) Oracle. However, the leader elected by the oracle is not unique, which is fundamentally different from consensus protocols that depend on the leader election.

VDF Verifiable Delay Functions (VDFs) [6] have been leveraged as a resource against Byzantine adversaries in various works [12, 29, 39, 57], 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.

CHAPTER 6

CONCLUSION

Sandglass and Gorilla show, for the first time, that it is possible to obtain consensus with deterministic safety in a permissionless model. This result suggests that it is the probabilistic nature of its PoW mechanism, rather that its permissionless model, that prevents deterministic safety in Nakamoto's consensus.

Furthermore, Gorilla 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. On the other hand, there exists a line of research [19, 41] focused on more practical solutions that can achieve deterministic safety in a dynamically available setting and reach decisions within a few rounds, albeit depending on a public key infrastructure (PKI). This opens up the possibility of developing a new protocol that combines the best of both approaches: a practical Byzantine fault-tolerant permissionless protocol that achieves deterministic safety but does not rely on a PKI.

APPENDIX A

CORRECTNESS OF SANDGLASS

We prove that Sandglass satisfies the consensus requirements. Figure A.1 illustrates the dependency between the statements proven below. The letters L, O, and C, signify Lemma, Observation, and Corollary, respectively.

Sandglass upholds the definitions of Valdity, Agreement, and Termination (with probability 1) given in Section 4.1.

A.1 Validity

We show that if all nodes have the same initial value, this is the only value that can be decided.

Lemma 9 (Validity)**.** *If all nodes that ever join the system have initial value v and any node (whether good or defective) decides, then it decides v.*

Proof. By line 22 of Sandglass, if a node p_i decides a value, it decides the value held in its variable v_i. By lines 2 and 12–15 of Sandglass, v_i is either the initial value of p_i, or one of the values that p_i receives. Therefore, it suffices to show that if all nodes have initial value v, then v is the only value that can be sent by any node.

We prove, by induction on the round number, that any message m sent by any node for round r proposes v.

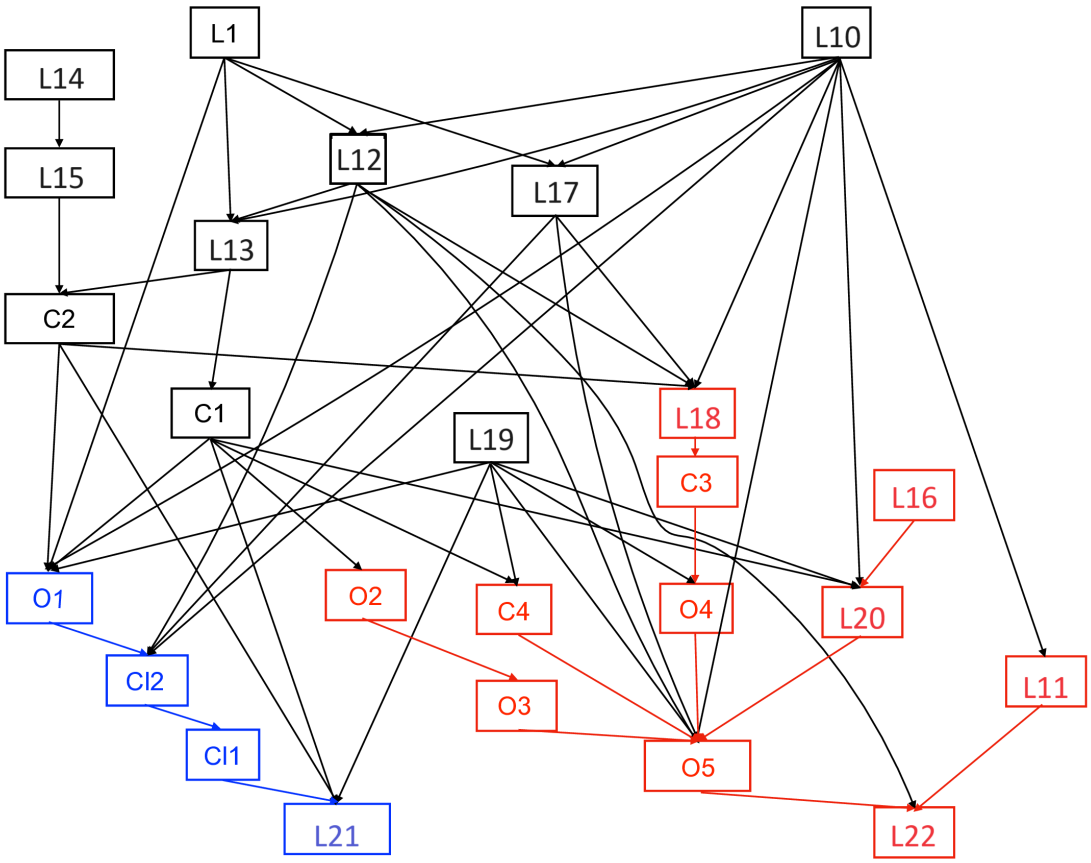
Base case: $r = 1$ Consider any node p that sends a message in the first round. Since every node's initial value is v, the message that p broadcasts at

line 25 proposes v.

Induction hypothesis: Assume that all messages sent by any node up to round $r = k$ propose v.

Induction step: Consider any node p sending a message in round $r = (k + 1)$ at step T. By assumption, all round k messages collected by p must be proposing v. In lines 12–15 of Sandglass, v_i is randomly selected from among the proposed values with highest priority collected in round k. Since the only collected value is v, v_i can only be set to v. □

Figure A.1: The structure of the proof is illustrated through the dependencies among its constituent lemmas, corollaries, claims, and observations. Preparatory results discussed in the Scaffolding section are shown in black; red and blue denote facts used in the proofs of Agreement and Termination, respectively.



A.2 Scaffolding

Before addressing Agreement and Termination, we prove several statements that will serve as scaffolding for our main results.

We start with some terminology. In Sandglass, a node in round r ignores every message it receives that was sent with some round $r' < (r-1)$ (line 9). We say that a node p *collects* message m if it adds it to M_p (line 10). We say that a node is in round r at step T if it sends a message for round r at step T.

Our first lemma establishes that good nodes are progressing almost together from round to round.

Lemma 10. *If a good node is in round r at step T, then all good nodes will be in round r or larger at step $(T+1)$.*

Proof. Let p be a good node in round r at step T.

By line 6 of Sandglass, p must have collected at least $\mathcal{T}$ messages for round $(r-1)$, which p will then forward to all nodes in the coffer of the message m that p broadcasts at line 25.

Consider any good node p' that is in a round $r' < r$ at T or joins the system at step $(T+1)$. Since p and p' are good, p' will receive m by $(T+1)$ and, by line 5, add to its set $Rec_{p'}$ both m and all the messages p forwarded in m's coffer, including at least $\mathcal{T}$ messages for round $(r-1)$. Then, computing at line 6 the largest round for which p' has received $\mathcal{T}$ messages or more will return at least $(r-1)$, and, at line 7, p' will update its round number, if it was smaller, to be at least r.

□

The next lemma establishes that good nodes progress by at least one round every $\mathcal{T}$ steps.

Lemma 11. *If, at step T, r is the earliest round that any good node is in, then at step $(T + \mathcal{T})$ all good nodes are at least in round $(r + 1)$.*

Proof. Let p be a good node that, by hypothesis, is in round r at step T; by assumption, all good nodes are at least in round r at step T. By Lemma 10, all good nodes are in round r or larger at step $(T+1)$; indeed, by a similar argument, all good nodes are in round r or larger for any step $T' > T$. Further, in each of these steps the system contains at least one good node, since, by assumption, the system contains at least one node in every step and a majority of its nodes are good.

For the time interval from T to $(T + \mathcal{T} - 1)$, consider all the good nodes in each of the steps of the interval. There are two cases:

- In some step of the interval, some good node is in some round $r' > r$.

  If so, by the same reasoning used above, all good nodes will be in round $r' \geq (r + 1)$ or larger at step $(T + \mathcal{T})$.

- In all steps of the interval, all good nodes are in round r.

  If so, in each of these steps there exists at least one good node that broadcasts a message for round r (by line 25 of Sandglass). Consider any good node p_g at step $(T + \mathcal{T})$. Again, there are two cases:

  - p_g receives some message m' for round $r' \geq (r + 1)$, from a defective node.

If so, by line 5, Rec_{p_g} will include at least the $\mathcal{T}$ messages for round $(r' - 1) \geq r$ forwarded on m'.

– p_g only receives messages for round r or smaller.

If so, p_g receives at least $\mathcal{T}$ messages for round r.

In both cases, computing at line 6 the largest round for which p_g has received $\mathcal{T}$ messages or more will return at least r, and, at line 7, p_g will update its round number, if it was smaller, to be at least $(r + 1)$.

□

Lemma 12. *At any step T, any defective node is at most one round ahead of any good node.*

Proof. By contradiction. Assume that there exists an earliest step, T, where some defective node p is more than one round ahead of a good node p_g, *i.e.*, at T node p is in some round r and node p_g is in round $r_{p_g} \leq (r - 2)$.

Note that no good node is in round $(r - 1)$ or larger before T; otherwise, by Lemma 10, all good nodes would be in round $(r - 1)$ or larger at T, contradicting $r_{p_g} \leq (r - 2)$. Therefore, defective node p received no messages from good nodes for round $(r - 1)$ by T.

Consider the earliest step $T' \leq (T - 1)$ where some defective node is in round $(r - 1)$. Since T is the first step where some defective node is more than a round ahead of a good node, all good nodes must be in round $(r - 2)$ or larger at T'; but, as we just showed, no good node is in round $(r - 1)$ or larger before T. Therefore, all good nodes must be in round $(r - 2)$ from T' until T.

Consider the k consecutive steps from T' to $(T-1)$. Let the number of messages generated by good nodes and defective nodes in each step be, respectively, $g_1, ..., g_k$ and $d_1, ..., d_k$. Since up to and including step T node p has received for round $(r-1)$ only messages from defective nodes, and yet p is in round r at T, by line 6 of Sandglass, $\Sigma_{i=1}^{i=k} d_i \geq \mathcal{T}$ and thus, by Lemma 1, $\Sigma_{i=2}^{i=k-1} g_i \geq \mathcal{T}$. Since by assumption every step includes at least one good node (*i.e.*, $g_1 > 0$), we have that $\Sigma_{i=1}^{i=k-1} g_i > \mathcal{T}$. Recall that during these k steps all good nodes are in round $(r-2)$; then, all messages $g_1, ..., g_k$ are for round $(r-2)$ and will all be received by all good nodes by T. By line 6 and line 7, then, *all* good nodes (including p_g) must be in round $(r-1)$ at T. This contradicts our assumption and completes the proof. □

Lemma 13. *For any r, a good node that enters round $(r+1)$ collects at least one message from a good node for round r.*

Proof. By contradiction. Let T be the first step where some good node p_g enters round $(r+1)$ without collecting any messages from any good node for round r.

Since, by line 9 of Sandglass, p_g collects all the messages it receives for round r, and yet it collects no messages from good nodes for round r, p_g must have received no messages for round r from good nodes by T.

By our model's assumptions about good nodes, this implies that no good node has sent a message for round r (and hence that no good node was in round r) before step T. Therefore, both of the following statements must be true:

S1: *Defective nodes generated at least $\mathcal{T}$ messages for round r before step T.* By line 6, a node must receive at least $\mathcal{T}$ messages for round r to be in

round $(r+1)$. Since p_g received no messages for round r from good nodes before T, all the messages p_g received for round r must be from defective nodes.

S2: *No good node moved past round* $(r-1)$ *before* T. We have showed above that no good node is in round r before T; further, since p_g is in round $(r+1)$ at T, by Lemma 10, no good node is in a round larger than $(r+1)$ before T. Finally, no good node p'_g can be in round $(r+1)$ at $T' < T$: otherwise, since good nodes send no messages for round r before T, p'_g would not have collected any message from a good node for round r at T', contradicting our assumption that T is the first step where some good node enters round $(r+1)$ without collecting any message from good nodes for round r. Hence, before T no good node can be in round r or larger: the largest round any good node can be in is round $(r-1)$.

Let T' be the earliest step when some defective node is in round r. By Lemma 12, the earliest round that any good node can be in at T' is round $(r-1)$. Combining this observation with S2, we conclude that all good nodes are in round $(r-1)$ from T' until T.

Denote by k the number of consecutive steps from T' to $(T-1)$. Let the number of messages generated by good nodes and defective nodes in each step be, respectively, $g_1, ..., g_k$ and $d_1, ..., d_k$. Since by T node p_g has received only messages from defective nodes for round r, and yet it is in round $(r+1)$ at T, then, by line 6 of Sandglass, $\Sigma_{i=1}^{i=k} d_i \geq \mathcal{T}$ and thus, by Lemma 1, $\Sigma_{i=1}^{i=k-1} g_i \geq \mathcal{T}$. Recall that during these k steps all good nodes are in round $(r-1)$; then, all messages $g_1, ..., g_{k-1}$ are for round $(r-1)$ and will all be received by all good nodes by $(T-1)$. By lines 6 and 7 of Sandglass, then, *all* good nodes (including

p_g) must be in round r at $(T-1)$, contradicting S2. □

It follows that all good nodes collect a message from a single good node for each round.

Corollary 1. *For any round r, there exists a message from a good node for round r that is collected by all good nodes that are in round* $r' \geq (r+1)$.

Proof. Consider any round r. Let T be the earliest step when some good node p_g reaches some round $(r+1)$. By Lemma 13, p_g collects by T at least one message, $m_{r,all}$, from a good node for round r. Since $m_{r,all}$ is sent by a good node, all good nodes must have received m by T.

Now we prove $m_{r,all}$ is collected by all good nodes that are in round $r' \geq (r+1)$ by induction on r'.

Base case: $r' = (r+1)$ We are going to prove that $m_{r,all}$ is collected by all good nodes that are in round $(r+1)$.

Since p_g is the earliest good node who reaches round $(r+1)$, any good node who reaches round $(r+1)$ at the same step or later must have collected $m_{r,all}$ at line 10 of Sandglass.

Induction hypothesis Assume $m_{r,all}$ is collected by all good nodes that are in round $r' = k \geq (r+1)$.

Induction step We are going to prove that the lemma holds for $r' = (k+1)$. By induction hypothesis, $m_{r,all}$ is collected by all good nodes that are in round k; i.e., $\forall m = (\cdot, \cdot, k, \cdot, \cdot, M)$ sent by a good node for round k, $m_{r,all} \in M$.

Consider the earliest step T when some good node p_g reaches round $(k+1)$.

By Lemma 13, p_g collects by T at least one message, $m_k = (\cdot, \cdot, k, \cdot, \cdot, M_k)$, from a good node for round k. As we argued above, $m_{r,all}$ must be included in M_k. Since m_k is sent by a good node, all good nodes must have received m by T. Since p_g is one of the earliest good node to reach round $(k+1)$, any good node that reaches round $(k+1)$ at the same step or later must have collected both m_k and all the messages in M_k, including $m_{r,all}$ at line 10 of Sandglass. Therefore, there exists a message from a good node for round r, namely $m_{r,all}$, that is collected by all the good nodes that are in round $(k+1)$. □

Lemma 14. *For any message $m = (\cdot, \cdot, r \geq 2, \cdot, \cdot, \cdot, M)$, M contains at least $\mathcal{T}$ messages generated for round $(r-1)$.*

Proof. Consider a message $m = (p, \cdot, r, \cdot, \cdot, \cdot, M_p)$ for any round $r \geq 2$.

Let T be the earliest step when p is in round r (i.e., the earliest step when p broadcasts at line 25 of Sandglass a message for round r). Independent of whether p has just been activated at T, or was already active in a round smaller than r at $(T-1)$, p's round number r_p must have been updated to r in line 7 of Sandglass. Therefore, the condition on line 6 must be satisfied, *i.e.*, Rec_p must contain at least $\mathcal{T}$ messages for round $(r-1)$ at T. Then by lines 9-10, M_p contains at least $\mathcal{T}$ messages for round $(r-1)$ at T. By line 24, any message p generates and broadcasts while in round r, either at T or later, contains at least $\mathcal{T}$ messages generated in round $(r-1)$. □

Lemma 15. *If a node p receives a message for round r at step T, then p will be in at least round r at step T.*

Proof. When p receives a message m generated in round r, it adds to Rec_p all the messages contained in the set M included in m (line 5 of Sandglass). By

Lemma 14, M contains at least $\mathcal{T}$ messages for round $(r-1)$; thus, if r_p is smaller than r at line 6, r_p will be set to at least r at line 7. □

Corollary 2. *At any step T, if a good node is in round $r \geq 2$, then any good node is at least in round $(r-1)$.*

Proof. Consider a good node p that is in round r at T. By Lemma 13, p must have collected at least one message, m, from a good node p_{r-1} for round $(r-1)$. Consider any other good node p', it must also have received m by T. By Lemma 15, p' is at least in round $(r-1)$ at step T. □

Lemma 16. *The round number of a node never decreases.*

Proof. The lemma follows trivially since line 5 of Sandglass only adds new messages to Rec_i; thus, the set of received messages used to compute the current round number at line 7 never shrinks. □

Lemma 17. *Let T_r and T_{r+1} be the earliest steps where all good nodes are, respectively, at least in rounds r and $(r+1)$. Let g_i and d_i denote, respectively, the number of messages generated by good nodes and defective nodes in the i-th step of the sequence of k steps starting from T_r and up to step $(T_{r+1}-1)$. Then, $\Sigma_{i=1}^{k} d_i < \mathcal{T}$.*

Proof. First of all, by Lemma 10, no good node is in a round smaller than r after T_r; and no good node is in a round smaller than $(r+1)$ after T_{r+1}.

If $k = 0$, *i.e.*, $T_r = T_{r+1}$, the lemma trivially holds.

If $k \geq 1$, it suffices to establish that $\Sigma_{i=1}^{k-1} g_i < \mathcal{T}$; then, by Lemma 1, we can conclude that $\Sigma_{i=1}^{k} d_i < \mathcal{T}$ and proves the lemma.

All that is left to prove then is that $\Sigma_{i=1}^{k-1} g_i < \mathcal{T}$ holds. To do so, we begin by observing that no good node is in round $(r+1)$ or later at step $(T_{r+1} - 2)$; otherwise, by Lemma 10, all good nodes would already be in round $(r+1)$ at step $(T_{r+1} - 1)$, *i.e.*, before T_{r+1}, which, by definition, is the earliest step where all good nodes are at least in round $(r+1)$.

Therefore, since all $\Sigma_{i=1}^{k-1} g_i$ messages sent by good nodes from T_r and up to $(T_{r+1} - 2)$ must be at least for round r, they must be exactly for round r.

From this, it immediately follows that $\Sigma_{i=1}^{k-1} g_i$ must be less that $\mathcal{T}$ (proving the lemma): if $\Sigma_{i=1}^{k-1} g_i$ equaled or exceeded $\mathcal{T}$, then all good nodes would have received at least $\mathcal{T}$ messages for round r by step $(T_{r+1} - 1)$ and thus would all be in round $(r+1)$ or larger at step $(T_{r+1} - 1)$, contradicting the definition of T_{r+1}. □

The following important lemma characterizes the rate of progress experienced by defective nodes that do not collect messages from good nodes. In particular, it establishes that defective nodes that do not collect any message from good nodes for $k\mathcal{T}$ consecutive rounds fall behind every good node by at least $(k-1)$ rounds.

Lemma 18. *Suppose a good node p_g is in round r at step T, and a node p_d is in round r_d at step $T' \leq T$. If p_d does not collect any messages from good nodes in any round $(r-i)$, where $0 \leq i < k\mathcal{T}$, then $r_d \leq (r - (k-1))$.*

Proof. To prove the lemma, we compute the maximum number of messages D_{max} that a defective node p_d can collect during the $k\mathcal{T}$ rounds when it does not collect any message from good nodes. To help us count these messages, for any $1 \leq i \leq k\mathcal{T}$, denote by $T_{(r-k\mathcal{T}+i)}$ the earliest step for which all good nodes are at least in round $(r - k\mathcal{T} + i)$.

Recall that, to be collected by p_d at step T', a message must have been generated no later than step $(T' - 1) \leq (T - 1)$. Then, we partition the execution of the system up to step $T - 1$ into three time intervals, and compute, for each interval, the maximum number of messages generated during these intervals that p_d could have collected for rounds $(r - k\mathcal{T} + 2)$ or larger.

I1: Up to step $(T_{(r-k\mathcal{T}+1)} - 1)$.

By definition of $T_{(r-k\mathcal{T}+1)}$, some good node is in some round $r' < r - k\mathcal{T} + 1$ at step $(T_{(r-k\mathcal{T}+1)} - 1)$. Therefore, neither a defective node nor a good node can be in some round $r'' > r - k\mathcal{T} + 1$ at step $(T_{(r-k\mathcal{T}+1)} - 1)$, respectively because of Lemma 12 and Corollary 2. Therefore, during this interval no messages were generated for rounds $(r - k\mathcal{T} + 2)$ or larger.

I2: From $T_{(r-k\mathcal{T}+1)}$ up to $(T_r - 1)$.

By assumption, p_d only collects messages generated by defective nodes throughout interval I2. We further split I2 into $(k\mathcal{T} - 1)$ consecutive subintervals, each going from $T_{(r-k\mathcal{T}+i)}$ up to $(T_{(r-k\mathcal{T}+i+1)} - 1)$ for $1 \leq i \leq (k\mathcal{T} - 1)$. By Lemma 17, in each of these sub-intervals defective nodes can generate at most $(\mathcal{T} - 1)$ messages. Therefore, the number of messages generated by defective nodes during I2 is at most $(\mathcal{T} - 1) \cdot (k\mathcal{T} - 1)$.

I3: Between T_r and $T - 1$.

Once again, by assumption p_d only collects messages generated by defective nodes throughout interval I3. There are two cases:

- $T - 1$ precedes T_r.

  If so, defective nodes trivially generate no messages during I3.

- $T - 1$ does not precede T_r.

By assumption, some good node p_g is in round r at T, where it collects all messages generated by good nodes before T; further, since p_g is still in round r, the messages for round r sent by good nodes before T must be fewer than $\mathcal{T}$. Finally, since p_g is in round r at T, by Lemma 10, in all steps preceding T no good node can be in round $(r+1)$ or higher. We then conclude that from step T_r and up to $(T-1)$ good nodes generated at most $(\mathcal{T}-1)$ messages, all for round r. Thus, since in any step defective nodes generate fewer messages than good nodes, during I3 defective nodes generate fewer than $(\mathcal{T}-1)$ messages.

Adding the messages generated in the three intervals, we find that D_{max}, the maximum number of messages that p_d could have collected up to and including step T for rounds $(r-k\mathcal{T}+2)$ or larger, is smaller than $(\mathcal{T}-1)\cdot k\mathcal{T}$; at the same time, since by assumption p_d is in round r_d, D_{max} must equal at least $(r_d-(r-k\mathcal{T}+2))\cdot\mathcal{T}$. Therefore, we have that $(r_d-(r-k\mathcal{T}+2))\cdot\mathcal{T} < (\mathcal{T}-1)\cdot k\mathcal{T}$, which implies $r_d \leq r-(k-1)$, proving the corollary. □

Corollary 3. *Suppose a good node p_g is in round r at step T, and a node p_d is in round $r_d=(r-1)$ at step $T' \leq T$. p_d must have collected some message from good nodes in some round r_g, where $r-3\mathcal{T} < r_g \leq (r-1)$.*

Proof. By contradiction. Assume that node p_d is in round $(r-1)$ and has not collected any message from good nodes in any round r_g, where $r-3\mathcal{T} < r_g < r$.

Note that by T' p_d has collected no message for round r as well, for, if it had, its round number would be at least r. To see why, suppose p_d collected m_r for round r. By line 5 of Sandglass, p_d would then add to its set Rec_{p_d} both m_r and

all the messages in the message coffer of m_r, including at least $\mathcal{T}$ messages for round $(r-1)$. Then, $\max_{|Rec_i(r)|\geq\mathcal{T}}(r)$ would be at least r at line 6, and p_d would update its round number to be at least r (line 7).

Therefore, p_d does not collect any message from good nodes in any round r_g, where $(r-3\mathcal{T}) < r_g \leq r$.

Then, by applying $k = 3$ in Lemma 18, $r_d \leq (r-2)$. Contradiction. □

The following lemma formalizes the semantics of the unanimity counter uC included in every message; it states that the value of uC in a message that proposes v is equal to the number of consecutive rounds in which the sender of m has collected only messages that propose v.

Lemma 19. *Consider a message* $m = (\cdot, \cdot, r, v, \cdot, uC > 0, M)$. *For any* $m' = (\cdot, \cdot, r', v', \cdot, uC', \cdot) \in M$, *where* $r - uC \leq r' < r$, *we have* $v' = v$ *and* $uC' \geq uC - (r - r')$.

Proof. By induction on uC.

Base case: $uC = 1$ We are going to prove that if a node broadcasts a message $m = (\cdot, \cdot, r, v, \cdot, uC = 1, M)$, then $\forall m' = (\cdot, \cdot, r', v', \cdot, uC', \cdot) \in M$, where $r' = (r-1)$, we have $v' = v$ and $uC' \geq uC - 1 = 0$.

Establishing that $uC' \geq 0$ follows trivially from the protocol. Since $r' = (r - 1) < r$, m' was added to M in line 10 of Sandglass (not in line 24). Note that Sandglass sets the value of $uCounter_i$ (at lines 16-19) only once per round, in the round's first step. Since by assumption the unanimity counter's value is 1, it must have been set at line 17; therefore, the condition at line 16 must be satisfied. Thus, for all m' in $M_i(r-1)$, v' equals the value v broadcast in m at line 25.

Induction hypothesis Assume the lemma holds for $uC = k > 0$.

Induction step We are going to prove that the lemma holds for $uC = (k+1)$.

First, we prove that for any $m_{r-1} = (\cdot, \cdot, r-1, v_{r-1}, \cdot, uC_{r-1}, \cdot) \in M$, it holds that $uC_{r-1} \geq uC - 1 = (k+1) - 1 = k$ and $v = v_{r-1}$. Since $uC = (k+1) > 0$, the value of the minimum value of the unanimity counter carried by all messages (including m_{r-1}) from round $(r-1)$ must be k; therefore, $uC_{r-1} \geq uC - 1 = k$. Finally, as in the Base Case, since *uCounter* is set at line 17 of Sandglass, the condition at line 16 is satisfied; therefore $v = v_{r-1}$.

Now, by line 10 of Sandglass, $\forall m' = (\cdot, \cdot, r', v', \cdot, uC', \cdot) \in M$, one of the following must be true:

Case 1 $r' = (r-1)$. It directly follows that $uC' \geq uC - 1$ and $v = v'$.

Case 2 There exists a message $m'' = (\cdot, \cdot, r'', v'', \cdot, uC'', M'') \in M$, where $r'' = (r-1)$ and $m' \in M''$. Since $r'' = (r-1)$, it follows again that $uC'' \geq uC - 1 = k$ and $v = v''$. Therefore, by the induction hypothesis, we have $\forall m_x = (\cdot, \cdot, r_x, v_x, \cdot, uC_x, \cdot) \in M''$, where $r'' - k \leq r_x < r''$, we have $v_x = v'' = v$ and $uC_x \geq uC'' - (r'' - r_x)$. Since $m' \in M''$, it follows that $v' = v$ and $uC' \geq uC'' - (r'' - r') \geq (uC - 1) - ((r-1) - r') \geq uC - (r - r')$.

□

Corollary 4. *If a good node p proposes v with uCounter $= uC$ for round r and $uC \geq 1$, then for any round r', where $r - uC \leq r' \leq r$, there exists a good node proposing v with uCounter at least $uC - (r - r')$.*

Proof. If a good node p proposes v with *uCounter* $= uC$ for round r and $uC \geq 1$, by Lemma 19, all the messages p collected for round r', where $r - uC \leq r' < r$

propose v with $uCounter \geq uC - (r - r')$. By Corollary 1, at least one of these messages is from a good node. Therefore, there exists a good node proposing v with $uCounter$ at least $uC - (r - r')$ for round r'.

For $r' = r$, the corollary trivially holds, since p proposes v with $uCounter = uC$ for round r. □

Lemma 20. *If a good node p sends a message proposing v with $uCounter > 0$ for round r, no good node sends a message proposing $v' \neq v$ with $uCounter > 0$ for round $(r - 1)$.*

Proof. By contradiction. Assume a good node p' sends a message, m', proposing $v' \neq v$ with $uCounter > 0$ for round $(r - 1)$.

Let T' be the first step when p' is in round $(r - 1)$, and let T be the first step when p is in round r.

Sandglass does not change the proposal value (v_i) or the priority counter ($uCounter_i$) during a round; therefore, p sends a message proposing v with $uCounter > 0$ for round r at T; and p' sends a message proposing v' with $uCounter > 0$ for round $(r - 1)$ at T'.

First, we are going to show that $T' = T$ by showing that neither $T' < T$ or $T' > T$ is possible.

Not $T' < T$ Assume $T' < T$. By model assumption, p will receive all the messages sent by good nodes on or before $T - 1$, which include m'. Since T is the first step where p is in round r, the condition in line 6 of Sandglass holds, and all the messages p received for round $(r - 1)$, including m', will

be collected by p at line 10 at T. By lines 16-19, since m' is proposing v', it is impossible for p to propose v with non-zero *uCounter*.

Not $T' > T$ Assume $T' > T$. Since p is in round r at T, then by Lemma 10, p' is at least in round r at $(T+1)$. Then, by Lemma 16, it is impossible for p' to be in round $(r-1)$ at $T' \geq (T+1)$.

Therefore, $T' = T$, *i.e.*, T is both the first step when p is in round r, and the first step when p' is in round $(r-1)$.

Now, we show that p' must have collected some message proposing v for round $(r-2)$.

By Corollary 1, there exists a message, $m_{all,r-1}$, from a good node for round $(r-1)$ that is collected by all the good nodes in round r, including p. We make two observations about $m_{all,r-1}$: (*i*) to be collected by T, $m_{all,r-1}$ must be sent before T; and (*ii*) since p proposes v with $uCounter > 0$, by Lemma 19, $m_{all,r-1}$ must propose v.

Let $m_{all,r-1} = (p_{all,r-1}, \cdot, r-1, v, \cdot, \cdot, M_{all,r-1})$. By lines 11-15 of Sandglass, v must be proposed by one of the round $(r-2)$ messages in $M_{all,r-1}$. Let one of the messages that propose v for round $(r-2)$ in $M_{all,r-1}$ be $m_{v,r-2}$.

Since p' is a good node, it must also have received $m_{all,r-1}$ by T. Therefore, by line 5 of Sandglass, all the messages in $M_{all,r-1}$, including $m_{v,r-2}$, are added to Rec' by p' at T.

Now, since we established that T is the first step when p' is in round $(r-1)$, p' updates its round number to $(r-1)$ at line 7 of Sandglass. Then, at line 10, p' collects all the round $(r-2)$ messages from the messages that it has received,

including $m_{v,r-2}$. Since $m_{v,r-2}$ proposes v, by lines 16-19, it is impossible for p' to propose v' with $uCounter > 0$ for round $(r-1)$. Contradiction. □

A.3 Agreement

Our strategy for proving Agreement (see Definition 4) proceeds in two phases and with the help of two claims, detailed below. We begin by *assuming* that Claim 1 holds, and rely on it to prove Agreement in Lemma 21. We do not prove Claim 1 directly, however: instead, we find it easier to prove Claim 2, which implies Claim 1, thus establishing Agreement.

Claim 1. *Let p_d be the earliest good node to decide, in round r_d at step T_d. Suppose p_d decides v_d. Then, any good node p_g that in any step (whether before, at, or after T_d) finds itself in a round r_g, where $r_g \geq r_d$, proposes v_d for r_g.*

We now prove that, assuming Claim 1 holds, so does Agreement.

Lemma 21 (Agreement)**.** *If a good node decides a value v, then no good node decides a value other than v.*

Proof. Denote by U^D the value of the unanimity counter at which a node decides. By lines 20 and 21 of Sandglass, $U^D = (6\mathcal{T} + 9)\mathcal{T}$. Let p_d be the first good node to decide; and suppose p_d decides v_d in round r_d at step T_d. By line 25, node p_d broadcasts at T_d a message $m_d = (p_d, \cdot, r_d, v_d, \cdot, uC_d)$, where $uC_d \geq U^D$.

By Lemma 19, all the messages p_d collected for any round $r_d - i$, $1 \leq i \leq U^D$, must be of the form $(\cdot, \cdot, r_d - i, v_d, \cdot, uCounter, \cdot)$, where $uCounter \geq U^D - i$.

By Corollary 2, at T_d no good node can be in a round earlier than $(r_d - 1)$; this implies, since p_d is the first good node to decide, that no good can decide prior to round $(r_d - 1)$.

We now show that it is impossible for any such node to decide on a value other than v_d – neither in $(r_d - 1)$, nor in r_d or in later rounds – thus proving the lemma.

Not in $(r_d - 1)$ By Corollary 1, there exists a message for round $(r_d - 2)$ broadcast by a good node that is collected by every good node that is in round $(r_d - 1)$ or larger. Let the message be m_{r_d-2}. Since m_{r_d-2} is also collected by p_d when it decides with $uCounter = U^D$, m_{r_d-2} is of the form $(\cdot, \cdot, r_d - 2, v_d, \cdot, uCounter_{r_d-2}, \cdot)$, where $uCounter_{r_d-2} \geq U^D - 2$. Consider any good node p_g in round $(r_d - 1)$. Since p_g collects m_{r_d-2} that proposes v_d in round $(r_d - 2)$, by lines 16-19 of Sandglass, the $uCounter$ of any value other than v_d proposed by p_g must be 0. Therefore, by lines 20-21, it is impossible for p_g to decide any value other than v_d in round $(r_d - 1)$.

Not in $r \geq r_d$ Trivially follows from Claim 1, any good node p_g in a round r, where $r \geq r_d$, will propose v_d, and cannot decide any value other than v_d.

□

Now we have shown if Claim 1 is true, Agreement is satisfied. To complete the proof, we proceed to show Claim 1 is true, and we are going to do it by proving the following claim that implies Claim 1. Claim 2 is at least as strong as Claim 1, since it adds to Claim 1 the additional requirement shown in bold.

Claim 2. *Let p_d be the earliest good node to decide, in round r_d at step T_d. Suppose p_d decides v_d. Then, any good node p_g that in any step (whether before, at, or after T_d)*

finds itself in a round r_g, where $r_g \geq r_d$, proposes v_d for r_g ***with priority at least 1****.*

Now, before proving Claim 2, we prove an observation that is useful to prove the claim.

Let U^D be the value of the unanimity counter at which a node decides. Since p_d decides at T_d, Sandglass requires p_d to broadcast at T_d a message $m_d = (p_d, \cdot, r_d, v_d, \cdot, uC_d)$, where $uC_d \geq U^D$; therefore, by Lemma 19, all the messages that p_d has collected for round r, where $r \geq r_d - U^D$, propose v_d, and their *uCounter* is at least $U^D - (r_d - r)$.

Definition 15 ((p_d, T_d)-D-form)**.** *Given a node p_d that decides v_d in round r_d at T_d, we say that a message for round r, where $r \geq r_d - U^D$, is* in (p_d, T_d)-D-form *if it proposes v_d and the uCounter is at least $U^D - (r_d - r)$. When p_d and T_d are clear from the context, we e say simply that the message is in* D-form*.*

It directly follows from Lemma 19 that all the messages that p_d collects from round $(r_d - U^D)$ to round $(r_d - 1)$ are in D-form.

Observation 1. *For any round r, where $r_d - U^D \leq r \leq r_d - 1$, let $T_r^{\forall}$ and $T_{r+1}^{\forall}$ be the earliest steps where all the good nodes are at least in round r and in round $(r+1)$, respectively. Consider the set that includes messages sent by defective nodes starting from $T_r^{\forall}$ and before $T_{r+1}^{\forall}$ and messages sent by good nodes for round r. The total number of messages not in* D-form *in this set is smaller than $\mathcal{T}$.*

Proof. Let $T_r^{\exists}$ be the earliest step when some good node is in round r. By Corollary 1, we know that $T_r^{\exists}$ exists for all r.

We are going to show that:

For any round r, where $r_d - U^D \leq r \leq r_d - 1$, all the messages sent by good nodes for round r before $T^{\exists}_{r+1}$ must be in D-form. (F1)

We prove two cases separately.

Case 1 $r_d - U^D \leq r \leq r_d - 2$. Consider the message, $m_{all,r+1}$, that all good nodes collect for round $(r+1)$. Since p_d also collects it, $m_{all,r+1}$ must be in D-form. Consider the node $p_{all,r+1}$ that sends $m_{all,r+1}$. By definition of $T^{\exists}_{r+1}$, $p_{all,r+1}$ enters round $(r+1)$ at or after $T^{\exists}_{r+1}$; therefore, $p_{all,r+1}$ must have collected all the messages sent by good nodes for round r before $T^{\exists}_{r+1}$. Since $m_{all,r+1}$ is in D-form, by Lemma 19, all the messages sent by good nodes for round r before $T^{\exists}_{r+1}$ must also be in D-form.

Case 2 $r = r_d - 1$, note that p_d decides in round r_d, and thus also sends a message in D-form for round r_d. Since p_d enters round r_d at or after $T^{\exists}_{r_d}$, p_d must have collected all the messages sent by good nodes for round $(r_d - 1)$ before $T^{\exists}_{r_d}$. Therefore, by Lemma 19, all messages sent by good nodes for round $r = r_d - 1$ before $T^{\exists}_{r+1}$ must also be in D-form.

Having established F1, we proceed to prove the observation.

By Corollary 2, all good nodes are at least in round r at $T^{\exists}_{r+1}$, i.e., good nodes are either in round r or in round $(r+1)$ at $T^{\exists}_{r+1}$. If all good nodes are in round $(r+1)$, then $T^{\forall}_{r+1} = T^{\exists}_{r+1}$; otherwise, by Lemma 10, $T^{\forall}_{r+1} = (T^{\exists}_{r+1} + 1)$. We consider these two cases separately.

Let S_r be the sequence of k steps starting from $T^{\forall}_r$ and up to $(T^{\forall}_{r+1} - 1)$ (perhaps $k = 1$). Let X equal the sum of (i) the number of messages sent by good

nodes for round r that are not in D-form, and (*ii*) the number of messages sent by defective nodes during S_r that are not in D-form. To prove Observation 1, it is sufficient to prove $X < \mathcal{T}$.

Let d_i denote the number of messages sent by defective nodes in the i-th step of S_r. Let g_i denote the number of messages sent by good nodes for round r in the i-th step of S_r.

Case 1 $T^{\forall}_{r+1} = T^{\exists}_{r+1}$. In this case, since all messages sent by good nodes for round r are sent before $T^{\exists}_{r+1}$, by F1, all messages sent by good nodes for round r are in D-form. Therefore, X is no more than the number of messages sent by defective nodes during S_r, i.e. $X \leq \Sigma_{i=1}^{k} d_i$.

Since, in this case, no good node is in round $(r+1)$ at $(T^{\forall}_{r+1} - 1)$, the number of messages sent by good nodes for round r before $(T^{\forall}_{r+1} - 1)$ is smaller than $\mathcal{T}$; otherwise, by line 6 of Sandglass, all good nodes would have proceeded to round $(r+1)$ at $(T^{\forall}_{r+1} - 1)$. Therefore, $\Sigma_{i=1}^{k-1} g_i < \mathcal{T}$. Then, by Lemma 1, $\Sigma_{i=1}^{k} d_i < \mathcal{T}$, therefore $X < \mathcal{T}$, an we are done.

Case 2 $T^{\forall}_{r+1} = T^{\exists}_{r+1} + 1$. Again, we proved in F1 that all messages sent by good nodes for round r before $T^{\exists}_{r+1}$ are in D-form. Therefore, X is no more than the sum of the messages sent by good node for round r at $T^{\exists}_{r+1}$ and the messages sent by defective nodes during S_r, i.e., $X \leq \Sigma_{i=1}^{k} d_i + g_k$.

Now we are going to show, using a set of inequalities, that $\Sigma_{i=1}^{k} d_k + g_k < \mathcal{T}$; $X < \mathcal{T}$ directly follows.

Let $\bar{d} = \frac{\Sigma_{i=1}^{k-1} d_i}{k-1}$, and $\bar{g} = \frac{\Sigma_{i=1}^{k-1} g_i}{k-1}$. Recall that, in all steps, good nodes outnumber defective nodes. Therefore, for all $1 \leq i \leq (k-1)$, we have $d_i \leq g_i - 1$. Then, for all $1 \leq i \leq (k-1)$, since $d_i \leq g_i - 1$ and $d_i + g_i \leq \mathcal{N}$, we have that $\bar{d} \leq \bar{g} - 1$ and $\bar{d} + \bar{g} \leq \mathcal{N}$. Dividing both inequalities by $\bar{g}$ yields

$\frac{\bar{d}}{\bar{g}} \leq \min(1 - \frac{1}{\bar{g}}, \frac{\mathcal{N}}{\bar{g}} - 1)$. Note that the largest value of $\min(1 - \frac{1}{\bar{g}}, \frac{\mathcal{N}}{\bar{g}} - 1)$ occurs when $1 - \frac{1}{\bar{g}} = \frac{\mathcal{N}}{\bar{g}} - 1$; solving for $\bar{g}$ and plugging the solution back in gives us: $\min(1 - \frac{1}{\bar{g}}, \frac{\mathcal{N}}{\bar{g}} - 1) \leq (1 - \frac{2}{\mathcal{N}+1})$. Therefore, we have $\frac{\bar{d}}{\bar{g}} \leq (1 - \frac{2}{\mathcal{N}+1})$, and thus

$$\bar{d} \leq \bar{g} \cdot (1 - \frac{2}{\mathcal{N}+1}). \tag{A.1}$$

Since $T^{\forall}_{r+1} = T^{\exists}_{r+1} + 1$, some good node is still in round r at $T^{\exists}_{r+1}$; therefore, the number of messages sent by good nodes for round r before $T^{\exists}_{r+1}$ is smaller than $\mathcal{T}$; otherwise, by line 6 of Sandglass, all good nodes would have proceeded to round $(r+1)$ at $T^{\exists}_{r+1}$. Therefore, $\Sigma_{i=1}^{k-1} g_i < \mathcal{T}$, i.e.,

$$\bar{g} \cdot (k-1) < \mathcal{T}. \tag{A.2}$$

Since at least one good node is already in round $(r+1)$ at $T^{\exists}_{r+1}$, the number of good nodes in round r plus the number of defective nodes at $T^{\exists}_{r+1}$ is no more than $\mathcal{N} - 1$, i.e.,

$$g_k + d_k \leq \mathcal{N} - 1. \tag{A.3}$$

Since good nodes outnumber defective nodes in all steps, we have for all $1 \leq i \leq (k-1)$

$$d_i \leq \frac{\mathcal{N} - 1}{2}. \tag{A.4}$$

Now we will show $(\Sigma_{i=1}^{k} d_i + g_k) < \mathcal{T}$.

$$
\begin{aligned}
(\Sigma_{i=1}^{k} d_i + g_k) &= d_k + g_k + (k-1)\bar{d} \\
&\leq (\mathcal{N}-1) + (k-1)\bar{d} && \text{(By Inequality A.3)} \\
&\leq (\mathcal{N}-1) + (k-1)\cdot \bar{g} \cdot (1 - \frac{2}{N+1}) && \text{(By Inequality A.1)} \\
&< (\mathcal{N}-1) + \mathcal{T} \cdot (1 - \frac{2}{\mathcal{N}+1}) && \text{(By Inequality A.2)} \\
&= (\mathcal{N}-1) + \mathcal{T} - \frac{2}{\mathcal{N}+1} \cdot \mathcal{T} \\
&\leq \mathcal{T} - \frac{2}{\mathcal{N}+1} \cdot \frac{\mathcal{N}^2}{2} + (\mathcal{N}-1) && \text{(Since } \mathcal{T} = \lceil \tfrac{\mathcal{N}^2}{2} \rceil \geq \tfrac{\mathcal{N}^2}{2}) \\
&= \mathcal{T} - \frac{1}{\mathcal{N}+1} < \mathcal{T}
\end{aligned}
$$

This concludes the second case and thus the proof. □

We can now proceed to prove Claim 2.

Proof. We are going to prove that, for any step T, if at T a good node p_g is in round $r_g \geq r_d$, then p_g proposes v_d with *priority* at least 1.

Let $U^1 = 6\mathcal{T}$ be the *uCounter* value, such that if *uCounter* is greater or equals to U^1, then *priority* is at least 1. Thus $U^D = (U^1 + 3)\mathcal{T} + U^1$ is the *uCounter* value that, once reached, allows a node to decide (lines 21-22 of Sandglass).

As the first step of our proof, we establish the following fact:

If a good node is in r_g at T, a node that, before T, proposes $v' \neq v_d$, can be at most in round $(r_g - U^1 - 1)$. (F2)

Assuming F2 holds, Claim 2 follows easily. Since by F2 all nodes that propose in round $(r_g - U^1)$ before T must propose v_d, then, by line 17 of Sandglass,

all nodes that propose in round $(r_g - U^1 + 1)$ before T must propose v_d with *uCounter* at least 1. A simple inductive argument then shows that all nodes that ever propose in round $(r_g - U^1 + i)$ before T, where $1 \leq i < U^1$, propose v_d with *uCounter* at least i. With $i = U^1 - 1$, messages sent for $r_{v'}$ before T must propose v_d with *uCounter* at least $(U^1 - 1)$; therefore, p_g must propose v_d with *uCounter* at least U^1 at T, i.e., with *priority* at least 1.

Before, proving F2, we introduce a useful notion: For each message m sent in round r, we consider the set of messages collected by the sender of m in round $(r-1)$; we call this set m's *bag* for round $(r-1)$.

Consider some node p' that sends $m^{v'}_{r'}$ proposing v' for round r'. By line 12 of Sandglass, p' must have collected for round $(r'-1)$ a message $m^{v'}_{r'-1}$ proposing v', whose *priority* was the largest among all messages in $m^{v'}_{r'}$'s *bag*. Inductively, consider message $m^{v'}_{r'-i}$: it must in turn contain in its bag a message $m^{v'}_{r'-i-1}$ proposing v', whose *priority* is the largest among all the messages in the bag. Therefore, there exists a chain of messages extending from round 1 to round r', where each of these messages proposes v'.

Consider these messages' bags. By construction of the chain, there exists exactly one bag per round, and at least one of the messages with the highest *priority* in each bag must be proposing v'.

Let $uCounter^{v'}_i$ be the value of *uCounter* of $m^{v'}_i$. By line 17 of Sandglass, for all $1 \leq i < r'$: $uCounter^{v'}_i \geq uCounter^{v'}_{i+1} - 1$. Therefore,

$$\text{for all } 1 \leq i < j \leq r' \text{: } uCounter^{v'}_i \geq uCounter^{v'}_j - (j - i). \tag{F3}$$

We now prove F2 by induction on the round number r_g that a good node, p_g, is in.

Induction Basis: $r_g = r_d$ Suppose a good node is in round r_g at T. We proceed by contradiction: assume that before T there exists a node, p', proposing v' in some round $r' > (r_g - U^1 - 1) = (r_d - U^1 - 1)$.

Proceeding as above, we construct the chain of messages for p' and consider the bags for every round from $(r_d-(U^D-U^1)-1)$ to (r_d-U^1-1). We will show that (i) at most one of these bags can contain messages in D-form; and (ii) the total number of messages not in D-form sent before T is not sufficient to fill these bags. Thus, it is impossible for a node that before T proposes $v' \neq v_d$ to advance up to round $(r_d - U^1)$, contradicting our assumption and proving the basis of the induction.

Proof of (i) We show that at most one of the bags for the rounds from $(r_d-(U^D-U^1)-1)$ to $(r_d - U^1 - 1)$ contains messages in D-form, i.e., messages that propose v_d in round r, where $r \geq r_d - U^D$, with *uCounter* at least $U^D - (r_d - r)$.

By contradiction: assume more than one of the bags for the rounds from $(r_d-(U^D-U^1)-1)$ to (r_d-U^1-1) contains messages in D-form. Consider two such bags, for round r_1 and r_2 respectively, where $(r_d-(U^D-U^1)-1) \leq r_1 < r_2 \leq (r_d - U^1 - 1)$. Consider now any message $m^d_{r_1}$ in D-form contained in the bag for round r_1; $m^d_{r_1}$ proposes v_d with $uCounter^d_{r_1} \geq U^D - (r_d - r_1)$. Similarly, any message $m^d_{r_2}$ in D-form contained in the bag for round r_2 proposes v_d with $uCounter^d_{r_2} \geq U^D - (r_d - r_2)$.

Now, let us consider the messages $m^{v'}_{r_1+1}$ and $m^{v'}_{r_2}$ on the chain. We showed

above (F3) that $uCounter^{v'}_{r_1+1} \geq uCounter^{v'}_{r_2} - (r_2 - (r_1 + 1))$. Since there is a message in D-form that proposes $v_d \neq v'$ in the bag of round r_1, by line 17 of Sandglass, $uCounter^{v'}_{r_1+1} = 0$. Therefore, $uCounter^{v'}_{r_2} \leq uCounter^{v'}_{r_1+1} + (r_2 - (r_1 + 1)) = r_2 - (r_1 + 1) \leq r_2 - (r_d - (U^D - U^1)) = U^D - U^1 - (r_d - r_2)$. Then, by line 20 of Sandglass, $priority^{v'}_{r_2} \leq \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5)$.

Recall that $m^d_{r_2}$ proposes v_d with $uCounter^d_{r_2} \geq U^D - (r_d - r_2)$. Then,

$$\begin{aligned} priority^d_{r_2} &\geq \max(0, \lfloor \frac{U^D - (r_d - r_2)}{\mathcal{T}} \rfloor - 5) \\ &= \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2) + 6\mathcal{T}}{\mathcal{T}} \rfloor - 5) \qquad \text{(Since } U^1 = 6\mathcal{T}\text{)} \\ &= \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1). \end{aligned}$$

Since $r_2 > r_1 \geq (r_d - (U^D - U^1) - 1)$, i.e., $r_2 \geq (r_d - (U^D - U^1))$, we have $U^D - U^1 - (r_d - r_2) \geq 0$. Then, $\lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1 \geq 1$. Therefore, $priority^d_{r_2} \geq \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1 > \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5$ and $priority^d_{r_2} \geq 1 > 0$. Therefore, $priority^d_{r_2} > \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5) \geq priority^{v'}_{r_2}$.

Now, consider $m^{v'}_{r_2+1}$. It collects both $m^{v'}_{r_2}$ and $m^d_{r_2}$. Since $m^{v'}_{r_2+1}$ proposes v', $m^{v'}_{r_2}$ must be the message with the highest priority among the messages collected by $m^{v'}_{r_2+1}$ for round r_2. However, $priority^d_{r_2} > priority^{v'}_{r_2}$. Contradiction.

Proof of (*ii*) Now we established that at most one of the bags for the rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_d - U^1 - 1)$ contains messages in D-form. That is, among these $(U^D - 2U^1 + 1)$ bags, $(U^D - 2U^1)$ of them contain only messages that are not in D-form. We will call these *ND-bags*. Since the size of each bag is at least $\mathcal{T}$,

$$\text{ND-bags contain at least } \mathcal{T} \cdot (U^D - 2U^1) \text{ messages.} \qquad \text{(F4)}$$

Let us now compute the largest number of messages they can contain. ND-bags can only contain messages not in D-form in any round from $(r_d - (U^D - U^1) - 1)$ to $(r_d - U^1 - 1)$ sent by (A) good nodes; or (B) defective nodes.

Recall that $T^{\forall}_r$ is the earliest step where all good nodes are in round r.

By Lemma 12, $T^{\forall}_{r_d-(U^D-U^1)-2}$ is the earliest step where some defective node can be in round $(r_d - (U^D - U^1) - 1)$. Then, the messages covered by case (B) must have been sent between steps $T^{\forall}_{r_d-(U^D-U^1)-2}$ and $(T-1)$. We can then partition this range of steps into four consecutive subranges:

B1 : from $T^{\forall}_{r_d-(U^D-U^1)-2}$ to $(T^{\forall}_{r_d-(U^D-U^1)-1} - 1)$

B2 : from $T^{\forall}_{r_d-(U^D-U^1)-1}$ to $(T^{\forall}_{r_d-U^1} - 1)$

B3 : from $T^{\forall}_{r_d-U^1}$ to $(T^{\forall}_{r_d} - 1)$

B4 : from $T^{\forall}_{r_d}$ to $(T-1)$

We now count the total number of messages covered by cases A and $B1$ to $B4$.

B1 By Lemma 17, the number of messages in $B1$ is at most $(\mathcal{T} - 1)$.

A and B2 Consider, for any round r_b, where $(r_d - (U^D - U^1) - 1) \leq r_b \leq (r_d - U^1 - 1)$, the set of messages S_{r_b} obtained by adding (i) messages sent by defective nodes starting from $T^{\forall}_{r_b}$ and before $T^{\forall}_{r_b+1}$; and (ii) messages not in D-form sent by good nodes for round r_b. By Observation 1, S_{r_b} contains fewer than $\mathcal{T}$ messages. Thus, the set

$$\bigcup_{r_b=(r_d-(U^D-U^1)-1)}^{(r_d-U^1-1)} S_{r_b},$$

which contains all messages covered by cases A and $B2$, consists of no more than $(\mathcal{T} - 1) \cdot (U^D - 2U^1 + 1)$ messages.

B3 By Lemma 17, the number of messages sent by defective nodes in the time interval from $T_r^\forall$ to $(T_{r+1}^\forall - 1)$ is at most $(\mathcal{T} - 1)$. Since $B3$ contains U^1such intervals, the number of messages sent in $B3$ is at most $(\mathcal{T} - 1) \cdot U^1$.

B4 Note that p_g is still in round r_d at T, and that, by Lemma 10 and the definition of $T_{r_d}^\forall$, all good nodes are in round r_d from $T_{r_d}^\forall$ to $(T - 1)$. Therefore, the number of messages good nodes generate during B4 is smaller than $\mathcal{T}$; otherwise, all good nodes would be at least in round $(r_d + 1)$ at T. Since good nodes outnumber defective nodes in any step, it follows that the number of messages sent by defective nodes between $T_{r_d}^\forall$ and $(T - 1)$ is at most $(\mathcal{T} - 1)$.

Therefore, adding the number of messages in $B1$, A and $B2$, $B3$, and $B4$, ND-bags can contain no more than $(\mathcal{T} - 1) + (\mathcal{T} - 1) \cdot (U^D - 2U^1 + 1) + (\mathcal{T} - 1) \cdot U^1 + (\mathcal{T} - 1)$ messages, i.e.,

$$(\mathcal{T} - 1) \cdot (U^D - U^1 + 3). \tag{A.5}$$

Recall F4: ND-bags contain at least

$$\mathcal{T} \cdot (U^D - 2U^1) \text{ messages.} \tag{A.6}$$

Therefore, we have

$$(\mathcal{T} - 1) \cdot (U^D - U^1 + 3) \geq \mathcal{T} \cdot (U^D - 2U^1),$$

which we rewrite as $\mathcal{T} \cdot (U^D - U^1 + 3) - \mathcal{T} \cdot (U^D - 2U^1) \geq U^D - U^1 + 3$, and finally as $U^D \leq (U^1 + 3)\mathcal{T} + U^1 - 3$.

However, since $U^D = (U^1 + 3)\mathcal{T} + U^1$, we have a contradiction. Q.E.D.

Induction hypothesis: $r_d \leq r_g \leq r_d + k$ We assume that if a good node is in r_g, where $r_d \leq r_g \leq r_d + k$, at T, then a node that, before T, proposes $v' \neq v_d$ can be at most in round $(r_g - U^1 - 1)$. As we argued above, this is enough to easily show a version of Claim 2 limited to the case when $r_d \leq r_g \leq r_d + k$.

Induction step: $r_g = r_d + k + 1$ Suppose a good node is in round $r_g = r_d + k + 1$ at T, and that Claim 2 holds for any round r, where $r_d \leq r \leq r_d + k$. We will prove that if a good node is in r_g at T, then a node that, before T, proposes $v' \neq v_d$ can be at most in round $(r_g - U^1 - 1)$.

We will assume, by contradiction, that there exists a node p' that before T proposes v' in some round $r' > (r_g - U^1 - 1)$. We will consider the following two cases: (1) $(r_g - U^1 - 1) \leq r_d - 1$, i.e., $r_g \leq r_d + U^1$; and (2) $(r_g - U^1 - 1) > r_d - 1$, i.e., $r_g > r_d + U^1$.

Case 1: $r_g \leq r_d + U^1$ Proceeding as above, we construct the chain of messages for p' and consider the bags for every round from $(r_d - (U^D - U^1) - 1)$ to $(r_g - U^1 - 1)$. With $r_g \leq r_d + U^1$, we have $(r_g - U^1 - 1) \leq r_d - 1$.

We will show that (i) at most one of these bags can contain messages in D-form; and (ii) the total number of messages not in D-form sent before T is not sufficient to fill these bags. Thus, it is impossible for a node that before T proposes $v' \neq v_d$ to advance up to round $(r_g - U^1)$, contradicting our assumption. The proof for Case 1 is very similar to how we proved the induction basis; we present it in full for completeness.

Proof of (i) We will show that at most one of the bags for the rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_g - U^1 - 1)$ contains messages in D-form.

By contradiction: assume more than one of the bags for the rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_g - U^1 - 1)$ contains messages in D-form. Consider two such bags, for round r_1 and r_2 respectively, where $(r_d - (U^D - U^1) - 1) \leq r_1 < r_2 \leq (r_g - U^1 - 1) \leq (r_d - 1)$. Consider now any message $m^d_{r_1}$ in D-form contained in the bag for round r_1; $m^d_{r_1}$ proposes v_d with $uCounter^d_{r_1} \geq U^D - (r_d - r_1)$. Similarly, any message $m^d_{r_2}$ in D-form contained in the bag for round r_2 proposes v_d with $uCounter^d_{r_2} \geq U^D - (r_d - r_2)$.

Now, let us consider the messages $m^{v'}_{r_1+1}$ and $m^{v'}_{r_2}$ on the chain. We showed above (F3) that $uCounter^{v'}_{r_1+1} \geq uCounter^{v'}_{r_2} - (r_2 - (r_1 + 1))$. Since there is a message in D-form that proposes $v_d \neq v'$ in the bag of round r_1, by line 17 of Sandglass, $uCounter^{v'}_{r_1+1} = 0$. Therefore, $uCounter^{v'}_{r_2} \leq uCounter^{v'}_{r_1+1} + (r_2 - (r_1 + 1)) = r_2 - (r_1 + 1) \leq r_2 - (r_d - (U^D - U^1)) = U^D - U^1 - (r_d - r_2)$. Then, by line 20 of Sandglass, $priority^{v'}_{r_2} \leq \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5)$.

Recall that $m^d_{r_2}$ proposes v_d with $uCounter^d_{r_2} \geq U^D - (r_d - r_2)$. Then,

$$
\begin{aligned}
priority^d_{r_2} &\geq \max(0, \lfloor \frac{U^D - (r_d - r_2)}{\mathcal{T}} \rfloor - 5) \\
&= \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2) + 6\mathcal{T}}{\mathcal{T}} \rfloor - 5) \quad \text{(Since } U^1 = 6\mathcal{T}\text{)} \\
&= \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1).
\end{aligned}
$$

Since $r_2 > r_1 \geq (r_d - (U^D - U^1) - 1)$, i.e., $r_2 \geq (r_d - (U^D - U^1))$, we have $U^D - U^1 - (r_d - r_2) \geq 0$. Then, $\lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1 \geq 1$. Therefore, $priority^d_{r_2} \geq \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1 > \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5$ and $priority^d_{r_2} \geq 1 > 0$. Therefore, $priority^d_{r_2} > \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5) \geq priority^{v'}_{r_2}$.

Now, consider $m^{v'}_{r_2+1}$. It collects both $m^{v'}_{r_2}$ and $m^d_{r_2}$. Since $m^{v'}_{r_2+1}$

proposes v', $m^{v'}_{r_2}$ must be the message with the highest priority among the messages collected by $m^{v'}_{r_2+1}$ for round r_2. However, $priority^{d}_{r_2} > priority^{v'}_{r_2}$. Contradiction.

Proof of (*ii*) Now we established that at most one of the bags for the rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_g - U^1 - 1)$ contains messages in D-form. That is, among these $(U^D - 2U^1 + (r_g - r_d) + 1)$ bags, $(U^D - 2U^1 + (r_g - r_d))$ of them contain only messages that are not in D-form. We will call these *ND-bags*. Since the size of each bag is at least $\mathcal{T}$,

$$\text{ND-bags contain at least } \mathcal{T} \cdot (U^D - 2U^1 + (r_g - r_d)) \text{ messages.} \quad \text{(F5)}$$

Let us now compute the largest number of messages they can contain. ND-bags can only contain messages not in D-form in any round from $(r_d - (U^D - U^1) - 1)$ to $(r_g - U^1 - 1)$ sent by (A) good nodes; or (B) defective nodes.

By Lemma 12, $T^{\forall}_{r_d-(U^D-U^1)-2}$ is the earliest step where some defective node can be in round $(r_d-(U^D-U^1)-1)$. Then, the messages covered by case (B) must have been sent from step $T^{\forall}_{r_d-(U^D-U^1)-2}$ to step $(T-1)$. We can then partition this range of steps into four consecutive subranges:

B1 : from $T^{\forall}_{r_d-(U^D-U^1)-2}$ to $(T^{\forall}_{r_d-(U^D-U^1)-1} - 1)$

B2 : from $T^{\forall}_{r_d-(U^D-U^1)-1}$ to $(T^{\forall}_{r_g-U^1} - 1)$

B3 : from $T^{\forall}_{r_g-U^1}$ to $(T^{\forall}_{r_g} - 1)$

B4 : from $T^{\forall}_{r_g}$ to $(T - 1)$

We now count the total number of messages covered by cases A and $B1$ to $B4$.

B1 By Lemma 17, the number of messages sent in $B1$ is at most $(\mathcal{T}-1)$.

A and B2 Consider, for any round r_b, where $(r_d-(U^D-U^1)-1) \leq r_b \leq (r_g - U^1 - 1)$, the set of messages S_{r_b} obtained by adding (i) messages sent by defective nodes starting from $T^{\forall}_{r_b}$ and before $T^{\forall}_{r_b+1}$; and (ii) messages not in D-form sent by good nodes for round r_b. By Observation 1, S_{r_b} contains fewer than $\mathcal{T}$ messages. Thus, the set

$$\bigcup_{r_b=(r_g-(U^D-U^1)-1)}^{(r_d-U^1-1)} S_{r_b},$$

which contains all messages covered by cases A and $B2$, consists of no more than $(\mathcal{T}-1)\cdot(U^D-2U^1+(r_g-r_d)+1)$.

B3 By Lemma 17, the number of messages sent by defective nodes in the time interval from $T^{\forall}_r$ to $(T^{\forall}_{r+1}-1)$ is at most $(\mathcal{T}-1)$. Since $B3$ contains U^1such intervals, the number of messages sent in $B3$ is at most $(\mathcal{T}-1)\cdot U^1$.

B4 Note that p_g is still in round r_g at T, and that, by Lemma 10 and the definition of $T^{\forall}_{r_g}$, all good nodes are in round r_g from $T^{\forall}_{r_g}$ to $(T-1)$. Therefore, the number of messages good nodes generate during $B4$ is smaller than $\mathcal{T}$; otherwise, all good nodes would be at least in round (r_g+1) at T. Since good nodes outnumber defective nodes in any step, it follows that the number of messages sent by defective nodes between $T^{\forall}_{r_g}$ and $(T-1)$ is at most $(\mathcal{T}-1)$.

Therefore, adding the number of messages in $B1$, A and $B2$, $B3$, and $B4$, ND-bags can contain no more than $(\mathcal{T}-1)+(\mathcal{T}-1)\cdot(U^D-2U^1+(r_g-r_d)+1)+(\mathcal{T}-1)\cdot U^1+(\mathcal{T}-1)$ messages, i.e.,

$$(\mathcal{T}-1)\cdot(U^D-U^1+(r_g-r_d)+3). \tag{A.7}$$

Recall F5: ND-bags contain at least

$$\mathcal{T} \cdot (U^D - 2U^1 + (r_g - r_d)) \text{ messages.} \tag{A.8}$$

Therefore, we have

$$\begin{aligned}
&(\mathcal{T}-1)\cdot(U^D - U^1 + (r_g - r_d) + 3) \geq \mathcal{T}\cdot(U^D - 2U^1 + (r_g - r_d)) \\
\Rightarrow\ &(\mathcal{T}-1)\cdot(U^D - U^1 + (r_g - r_d) + 3) \geq (\mathcal{T}-1)\cdot(U^D - 2U^1 + (r_g - r_d)) \\
&\qquad\qquad + (U^D - 2U^1 + (r_g - r_d)) \\
&\Rightarrow (\mathcal{T}-1)\cdot(U^1 + 3) \geq U^D - 2U^1 + (r_g - r_d) \\
&\Rightarrow (\mathcal{T}-1)\cdot(U^1 + 3) \geq (U^1+3)\mathcal{T} - U^1 + (r_g - r_d) \\
&\qquad\qquad \text{(since } U^D = (U^1+3)\mathcal{T} + U^1\text{)} \\
&\Rightarrow 0 \geq 3 + (r_g - r_d)
\end{aligned}$$

However, since $r_g \geq r_d$, we have a contradiction. Q.E.D.

Case 2: $r_g > r_d + U^1$ Again, we construct the chain of messages for p' and consider the bags for every round from $(r_d - (U^D - U^1) - 1)$ to $(r_g - U^1 - 1)$.

We will show that:

(*i*) At most one of the bags for rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_d - 1)$ contains messages in D-form. That is, among these $(U^D - U^1 + 1)$ bags, $(U^D - U^1)$ of them contain only messages that are not in D-form. We will call these *ND-bags*.

(*ii*) Among the bags for rounds from r_d to $(r_g - U^1 - 1)$, at most one in every U^1 bags can contain messages from good nodes. That is, among these $(r_g - r_d - U^1)$ bags, $(r_g - r_d - U^1 - \lceil \frac{r_g - r_d - U^1}{U^1} \rceil)$ of them contain only messages from defective nodes. We will call these *Def-bags*.

(*iii*) The sum of (1) the messages not in D-form for round $(r_d-(U^D-U^1)-1)$ to (r_d-1), and (2) the messages sent by defective nodes for round r_d to (r_g-U^1-1) before T, is not sufficient to fill ND-bags and Def-bags.

Thus, it is impossible for a node that before T proposes $v' \neq v_d$ to advance up to round $(r_g - U^1)$, contradicting our assumption.

Proof of (*i*) By contradiction: assume more than one of the bags for the rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_d - 1)$ contains messages in D-form. Consider two such bags, for round r_1 and r_2 respectively, where $(r_d - (U^D - U^1) - 1) \leq r_1 < r_2 \leq (r_d - 1)$. Consider now any message $m^d_{r_1}$ in D-form contained in the bag for round r_1; $m^d_{r_1}$ proposes v_d with $uCounter^d_{r_1} \geq U^D - (r_d - r_1)$. Similarly, any message $m^d_{r_2}$ in D-form contained in the bag for round r_2 proposes v_d with $uCounter^d_{r_2} \geq U^D - (r_d - r_2)$.

Now, let us consider the messages $m^{v'}_{r_1+1}$ and $m^{v'}_{r_2}$ on the chain. We showed above (F3) that $uCounter^{v'}_{r_1+1} \geq uCounter^{v'}_{r_2} - (r_2 - (r_1 + 1))$. Since there is a message in D-form that proposes $v_d \neq v'$ in the bag of round r_1, by line 17 of Sandglass, $uCounter^{v'}_{r_1+1} = 0$. Therefore, $uCounter^{v'}_{r_2} \leq uCounter^{v'}_{r_1+1} + (r_2 - (r_1 + 1)) = r_2 - (r_1 + 1) \leq r_2 - (r_d - (U^D - U^1)) = U^D - U^1 - (r_d - r_2)$. Then, by line 20 of Sandglass, $priority^{v'}_{r_2} \leq \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5)$.

Recall that $m^d_{r_2}$ proposes v_d with $uCounter^d_{r_2} \geq U^D - (r_d - r_2)$. Then,

$$
\begin{aligned}
priority^d_{r_2} &\geq \max(0, \lfloor \frac{U^D - (r_d - r_2)}{\mathcal{T}} \rfloor - 5) \\
&= \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2) + 6\mathcal{T}}{\mathcal{T}} \rfloor - 5) \quad \text{(Since } U^1 = 6\mathcal{T}\text{)} \\
&= \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1).
\end{aligned}
$$

Since $r_2 > r_1 \geq (r_d - (U^D - U^1) - 1)$, i.e., $r_2 \geq (r_d - (U^D - U^1))$, we have $U^D - U^1 - (r_d - r_2) \geq 0$. Then, $\lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1 \geq 1$. Therefore, $priority^d_{r_2} \geq \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor + 1 > \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5$ and $priority^d_{r_2} \geq 1 > 0$. Therefore, $priority^d_{r_2} > \max(0, \lfloor \frac{U^D - U^1 - (r_d - r_2)}{\mathcal{T}} \rfloor - 5) \geq priority^{v'}_{r_2}$.

Now, consider $m^{v'}_{r_2+1}$. It collects both $m^{v'}_{r_2}$ and $m^d_{r_2}$, and $m^{v'}_{r_2}$. Since $m^{v'}_{r_2+1}$ proposes v', $m^{v'}_{r_2}$ must be the message with the highest priority among the messages collected by $m^{v'}_{r_2+1}$ for round r_2. However, $priority^d_{r_2} > priority^{v'}_{r_2}$. Contradiction.

Proof of (*ii*) We will show that among the bags for rounds r_d to $(r_g - U^1 - 1)$, at most one in every U^1 bags can contain messages from good nodes. By contradiction: assume there exist two bags containing messages from good nodes, for round r_1 and r_2 respectively, where $r_d \leq r_1 < r_2 \leq (r_g - U^1 - 1)$, and $r_2 - r_1 < U^1$. Consider now any message $m^g_{r_1}$ from a good node contained in the bag for round r_1; by induction hypothesis, $m^g_{r_1}$ proposes v_d with $priority^g_{r_1} \geq 1$. Similarly, any message $m^g_{r_2}$ from a good node contained in the bag for round r_2 proposes v_d with $priority^g_{r_2} \geq 1$.

Now, let us consider messages $m^{v'}_{r_1+1}$ and $m^{v'}_{r_2}$ on the chain. We showed above (F3) that $uCounter^{v'}_{r_1+1} \geq uCounter^{v'}_{r_2} - (r_2 - (r_1 + 1))$. Since there is a message from a good node proposing $v_d \neq v'$ in the bag of round r_1, by line 17 of Sandglass, $uCounter^{v'}_{r_1+1} = 0$. Therefore, $uCounter^{v'}_{r_2} \leq uCounter^{v'}_{r_1+1} + (r_2 - (r_1 + 1)) = r_2 - r_1 - 1 < U^1$. Then, by line 20 of Sandglass, $priority^{v'}_{r_2} < 1$.

Recall that $m^g_{r_2}$ proposes v_d with $priority^g_{r_2} \geq 1$. Then, we have $priority^g_{r_2} > priority^{v'}_{r_2}$.

Now, consider $m^{v'}_{r_2+1}$. It collects both $m^{v'}_{r_2}$ and $m^g_{r_2}$. Since $m^{v'}_{r_2+1}$ proposes v', $m^{v'}_{r_2}$ must be the message with the highest priority among the messages collected by $m^{v'}_{r_2+1}$ for round r_2. However, $priority^g_{r_2} > priority^{v'}_{r_2}$. Contradiction.

Proof of (*iii*) Now we established that:

(*i*) Among the bags for rounds from $(r_d - (U^D - U^1) - 1)$ to $(r_d - 1)$, $(U^D - U^1)$ ND-bags contain only messages that are not in D-form.

(*ii*) Among the bags for rounds from r_d to $(r_g - U^1 - 1)$, $(r_g - r_d - U^1 - \lceil \frac{r_g - r_d - U^1}{U^1} \rceil)$ Def-bags contain only messages from defective nodes.

Since each bag contains at least $\mathcal{T}$ messages, ND-bags and Def-bags contain collectively at least

$$\mathcal{T} \cdot (U^D - U^1 + (r_g - r_d - U^1 - \lceil \frac{r_g - r_d - U^1}{U^1} \rceil)) \text{ messages.} \tag{F6}$$

Let us now compute the largest number of messages they can contain. ND-bags and Def-bags can contain messages not in D-form in any round from $(r_d - (U^D - U^1) - 1)$ to $(r_d - 1)$ sent by either (A) good nodes or (B) defective nodes, and (C) messages sent for round r_d to $r_g - U^1 - 1$ by defective nodes.

By Lemma 12, $T^{\forall}_{r_d-(U^D-U^1)-2}$ is the earliest step where some defective node can be in round $(r_d - (U^D - U^1) - 1)$. Then, the messages covered by case (B) and (C) must have been sent from step $T^{\forall}_{r_d-(U^D-U^1)-2}$ to step $(T - 1)$. We can then partition this range of steps into four

consecutive subranges:

BC1 : from $T^{\forall}_{r_d-(U^D-U^1)-2}$ to $(T^{\forall}_{r_d-(U^D-U^1)-1}-1)$

BC2 : from $T^{\forall}_{r_d-(U^D-U^1)-1}$ to $(T^{\forall}_{r_d}-1)$

BC3 : from $T^{\forall}_{r_d}$ to $(T^{\forall}_{r_g}-1)$

BC4 : from $T^{\forall}_{r_g}$ to $(T-1)$

We now count the total number of messages covered by cases A and $BC1$ to $BC4$.

BC1 By Lemma 17, the number of messages in $BC1$ is at most $(\mathcal{T}-1)$.

A and BC2 Consider, for any round r_b, where $(r_d-(U^D-U^1)-1) \leq r_b \leq (r_d-1)$, the set of messages S_{r_b} obtained by adding (i) messages sent by defective nodes starting from $T^{\forall}_{r_b}$ and before $T^{\forall}_{r_b+1}$; and (ii) messages not in D-form sent by good nodes for round r_b. By Observation 1, S_{r_b} contains fewer than $\mathcal{T}$ messages. Thus, the set

$$\bigcup_{r_b=(r_d-(U^D-U^1)-1)}^{(r_d-1)} S_{r_b},$$

which contains all messages covered by cases A and $BC2$, consists of no more than $(\mathcal{T}-1)\cdot(U^D-U^1+1)$ messages.

BC3 By Lemma 17, the number of messages sent by defective nodes in the time interval from $T^{\forall}_r$ to $(T^{\forall}_{r+1}-1)$ is at most $(\mathcal{T}-1)$. Since $B3$ contains (r_g-r_d) such intervals, the number of messages sent in $BC3$ is at most $(\mathcal{T}-1)\cdot(r_g-r_d)$.

BC4 Note that p_g is still in round r_g at T, and that, by Lemma 10 and the definition of $T^{\forall}_{r_g}$, all good nodes are in round r_g from $T^{\forall}_{r_g}$ to $(T-1)$. Therefore, the number of messages good nodes generate during $BC4$ is smaller than $\mathcal{T}$; otherwise, all good nodes

would be at least in round $(r_g + 1)$ at T. Since good nodes outnumber defective nodes in any step, it follows that the number of messages sent by defective nodes between $T^{\forall}_{r_g}$ and $(T-1)$ is at most $(\mathcal{T}-1)$.

Therefore, adding the number of messages in $BC1$, A and $BC2$, $BC3$, and $BC4$, ND-bags and Def-bags can contain no more than $(\mathcal{T}-1) + (\mathcal{T}-1)\cdot(U^D - U^1 + 1) + (\mathcal{T}-1)\cdot(r_g - r_d) + (\mathcal{T}-1)$ messages, i.e.,

$$(\mathcal{T}-1)\cdot(U^D - U^1 + (r_g - r_d) + 3). \tag{A.9}$$

Recall F6: ND-bags and Def-bags contain at least

$$\mathcal{T}\cdot(U^D - U^1 + (r_g - r_d - U^1 - \lceil\frac{r_g - r_d - U^1}{U^1}\rceil)) \text{ messages.} \tag{F6}$$

Therefore, we have

$$(\mathcal{T}-1)\cdot(U^D - U^1 + (r_g - r_d) + 3) \geq \mathcal{T}\cdot(U^D - U^1 + (r_g - r_d - U^1 - \lceil\frac{r_g - r_d - U^1}{U^1}\rceil))$$

$$\Rightarrow \mathcal{T}\cdot(U^1 + 3 + \lceil\frac{r_g - r_d - U^1}{U^1}\rceil) \geq U^D - U^1 + (r_g - r_d) + 3$$

$$\Rightarrow \mathcal{T}\cdot(U^1 + 3 + \lceil\frac{r_g - r_d - U^1}{U^1}\rceil) \geq (U^1 + 3)\mathcal{T} + (r_g - r_d) + 3$$

(Since $U^D = ((U^1 + 3)\mathcal{T} + U^1)$)

$$\Rightarrow \lceil\frac{r_g - r_d}{6\mathcal{T}}\rceil - 1 \geq \frac{(r_g - r_d) + 3}{\mathcal{T}}$$ (Since $U^1 = 6\mathcal{T}$)

$$\Rightarrow \frac{r_g - r_d}{6\mathcal{T}} > \frac{(r_g - r_d) + 3}{\mathcal{T}}$$

(Since $\frac{r_g - r_d}{6\mathcal{T}} > (\lceil\frac{r_g - r_d}{6\mathcal{T}}\rceil - 1)$)

However, since $0 < (r_g - r_d) < ((r_g - r_d) + 3)$ and $(6\mathcal{T}) > \mathcal{T} > 0$, we have a contradiction. Q.E.D.

This concludes our proof of Agreement. □

A.4 Termination

The Termination property requires good nodes that stay active to eventually decide. Sandglass's Termination guarantee is probabilistic: For Termination to hold, Sandglass needs to be lucky. To help us prove that luck befalls Sandglass with probability 1, we introduce the interdependent notions of *lucky period*, *lucky value*, and *lucky round*.

Intuitively, a lucky period is a sequence of steps that leads to a decision: all nodes that are active in the step that immediately follows the end of the lucky period are guaranteed to decide in that step, if not earlier. The quality that makes a period lucky is straightforward. Recall that in Sandglass, if a node receives distinct highest priority proposals in the previous round, it can choose uniformly at random among them which one it is going to propose in the current round. During a lucky period, all the random choices that occur in a given round just happen to select the same value – the *lucky value* for that round. We give below a simple rule that defines what constitutes the lucky value for any given round spanned by the lucky period. To prove that Sandglass guarantees Termination with probability 1, we will proceed in two steps. First, we will show that the unanimity counter of all good nodes that are active during the last step of a lucky period reaches a value that allows them to decide. Second, we will prove that lucky periods occur with non-zero probability. Since in any infinite execution lucky periods appear infinitely often, it follows that any good node that stays active, no matter when it joins, is guaranteed to eventually decide.

Lucky value The rule that determines the lucky value for a given round r is

defined in terms of two sets. The first, $C(r, p)$, is independently computed by every node p as the set of messages for round r defined by line 11 of Sandglass; it contains the highest-priority messages p collected for round $(r-1)$. The second set, $O(r)$, contains a (possibly empty) subset of good nodes, and is defined across all good nodes that enter round r at any time. It contains any good node p_g that meets the following two criteria: (1) p_g has collected exactly one highest priority value in round $(r-1)$ (which p_g is then required to propose in round r) and (2) one of the messages sent by p_g in round r is collected by all good nodes in round $(r+1)$. Note that if $O(r)$ contains multiple good nodes, they may differ in the single highest priority value they have collected.

We dub the first round of a lucky period a *lucky round*. The lucky value $v_\ell(r)$ for a given round r of a given lucky period is defined inductively, with the base case defined by that period's lucky round, r_{start}, as follows:

- When $r = r_{start}$:

  If $O(r_{start}) \neq \emptyset$ and $\forall p \in O(r_{start}), v \in C(r_{start}, p)$, then $v_\ell(r_{start}) = v$.

  Otherwise, $v_\ell(r_{start})$ is arbitrarily set to one of the initial values. We will assume, without loss of generality, that $v_\ell(r_{start})$ is set to a.

- When $r > r_{start}$:

  If $O(r) \neq \emptyset$ and $\forall p \in O(r), v \in C(r, p)$, then $v_\ell(r) = v$.

  Otherwise, $v_\ell(r) = v_\ell(r-1)$.

Lucky period. We already saw that, informally, a lucky round is the first round of a lucky period. To define these notions more precisely, we introduce the following definitions, which we will use extensively in our Termination proof:

- $T_1(r)$: The earliest step where some node, possibly defective, is in round r.
- r_{lock}: The round with index $(r + 6\mathcal{T})$. We will prove that, if r is a lucky round then, in *every round* from $(r_{lock} + 1)$ to the end of the lucky period, v_ℓ is the same as the lucky value of round r_{lock}, and all good nodes propose the lucky value of round r_{lock}.
- $T(r_{lock})$: The earliest step where some node is in round r_{lock}.
- $\overleftrightarrow{P_\ell}$: A constant, equal to $(6\mathcal{T} + \lceil \frac{(6\mathcal{T}-1)\cdot U^D + 18\mathcal{T}}{5} \rceil)$, which denotes the number of rounds spanned by a lucky period, i.e., all rounds from the period's lucky round r_{start} to round $(r_{start} + \overleftrightarrow{P_\ell} - 1)$ (or, equivalently, round $(r_{lock} + \lceil \frac{(6\mathcal{T}-1)\cdot U^D + 18\mathcal{T}}{5} \rceil - 1)$.
- $T_D(r)$: The earliest step where all good nodes are in round $(r + \overleftrightarrow{P_\ell})$ or later. We will prove that, if r is a lucky round, then all good nodes decide by step $T_D(r)$.

We then say that r_{start} is a *lucky round* if, in every step during the lucky period from $T_1(r_{start})$ to $(T_D(r_{start}) - 1)$, whenever the set $C(r, p)$ of a node p in round r (where $r_{start} \leq r < r_{start} + \overleftrightarrow{P_\ell}$) holds multiple values, p randomly chooses to propose that round's lucky value, i.e., $v_\ell(r)$.

We now prove two observations that are useful for the proof for termination.

Observation 2. *Suppose r_{start} is lucky and consider round r, where $r_{start} \leq r < r_{start} + \overleftrightarrow{P_\ell}$. If $v_\ell(r) = v$, then all good nodes in round $(r + 1)$ collect at least one message proposing v for round r.*

Proof. By contradiction. Assume $v_\ell(r) = v$ and that some good node in round $(r+1)$ does not collect v for round r.

Let $A(r)$ be the set of good nodes whose messages for round r are collected by all the good nodes in round $(r+1)$. By Corollary 1, $A(r) \neq \emptyset$. Since some good node does not collect v for round r, it follows that none of the good nodes in $A(r)$ proposes v for round r, i.e. all the good nodes in $A(r)$ propose $v' \neq v$ for round r.

Since $v_\ell(r) = v$, for any node p, if $v \in C(r, p)$, then p must propose v for round r. Note that all the good nodes in $A(r)$ propose v' for round r, therefore, for any $p_g \in A(r)$, $C(r, p_g)$ only contains v'. That is, $O(r) = A(r) \neq \emptyset$. By definition of v_ℓ, $v_\ell(r)$ should be set to v'. Contradiction. □

Observation 3 (Necessary condition for v_ℓ flipping)**.** *If r_{start} is lucky, then for any r where $r_{start} < r < r_{start} + \vec{P}_\ell$, $v_\ell(r)$ is different from $v_\ell(r-1)$ only if some good node collects from round $(r-1)$ some message proposing $v_\ell(r)$ with priority at least* 1.

Proof. Assume $v_\ell(r-1) = v'$ and $v_\ell(r) = v$, where $v' \neq v$.

Since $r > r_{start}$ and $v' \neq v$, by definition of v_ℓ, $O(r) \neq \emptyset$ and for any $p \in O(r)$, $v \in C(r, p)$. Now we consider the values that one such good node, $p_g \in O(r)$, collects from round $(r-1)$. By Observation 2, p_g collects at least one message proposing v' from round $(r-1)$. However, only v is in $C(r, p_g)$. Therefore, p must have collected a message proposing v with a higher *priority* than v', that is, at least 1.

□

Observation 4. *If some good node in round r at T collects from round $(r-1)$ some message m proposing v with priority at least* 1*, then there exists a good node proposing v*

with uCounter larger than $3\mathcal{T}$ in round r_g, where $r - 3\mathcal{T} < r_g \leq r - 1$.

Proof. Consider the node p that at step $T' < T$ sends m, which proposes v with *priority* at least 1 for round $(r-1)$. By Corollary 3, p must have collected a message from a good node m_g by T' for round r_g, where $r - 3\mathcal{T} < r_g \leq r - 1$. Since p sends m with *priority* 1, *i.e.* $uCounter_p \geq 6\mathcal{T}$; then, by Lemma 19, m_g must propose v' with $uCounter_g \geq uCounter_p - ((r-1) - r_g) \geq 6\mathcal{T} - ((r-1) - r_g) > 3\mathcal{T}$. □

Observation 5. *If round r_{start} is a lucky round, then all good nodes active at step $T_D(r_{start})$ have decided by $T_D(r_{start})$.*

Proof. Recall that $T(r_{lock})$ is the earliest step where some node is in round r_{lock}, and $r_{lock} = r_{start} + 6\mathcal{T}$. Let $v_\ell(r_{lock})$ be v.

The proof proceeds in two main steps. In Step 1, we will prove that:

For any r, where $r_{lock} < r \leq r_{start} + \overleftrightarrow{P}_\ell - 1$, $v_\ell(r) = v_\ell(r_{lock}) = v$, and all good nodes propose v for round r. (F7)

In Step 2, relying on F7, we are going to prove that, for any good node p_g, the *uCounter* of $v = v_\ell(r_{lock})$ at $T_D(r_{start})$ will be at least U^D, upon which p_g will decide v.

Step 1 To prove F7, we are going to prove:

No good node in round r, where $r_{lock} \leq r \leq r_{start} + \overleftrightarrow{P}_\ell - 1$, collects a message proposing $v' \neq v$ for round $(r-1)$ with *priority* larger than 0. (F8)

Assuming F8 is true, then it is easy to show F7 is true as follows. By combining F8 and Observation 3, we can conclude that $v_\ell(r_{lock})$ is the lucky

value for all rounds from r_{lock} to $(r_{start} + \overleftrightarrow{P}_\ell - 1)$. Now, consider any round r, where $r_{lock} + 1 \leq r \leq r_{start} + \overleftrightarrow{P}_\ell - 1$. By Observation 2, since $v_\ell(r-1) = v$, all good nodes in round r collect at least one message proposing v for round $(r-1)$. By F8, we know that no good node in round r collects a message proposing v' for round $(r-1)$ with *priority* larger than 0 . Therefore, any good node in round r either collects only v, or collects both v and v', where the *priority* of v' is 0. Since r_{start} is a lucky round, all good nodes propose v for round r, proving F7.

We are going to prove F8 by contradiction. Let r' be the earliest round in the range from r_{lock} to $(r_{start} + \overleftrightarrow{P}_\ell - 1)$, where some good node, currently in r', collects a message from round $(r'-1)$ proposing $v' \neq v$ with *priority* at least 1.

We are going to prove that (i) there exists a round r_g, where $(r_{lock} - 3\mathcal{T}) < r_g \leq (r'-1)$, such that (a) a good node p_g proposes v' in round r_g with $uCounter_g > 3\mathcal{T}$, and (b) $v_\ell(r_g - 1) = v'$; and (ii) $(r_g - 1)$ can be neither smaller nor larger than r_{lock}. Since $(r_g - 1) \neq r_{lock}$, as their v_ℓ values are different, this leads to a contradiction.

Proof of (i) Since some good node in round r' at T collects a message proposing v' for round $(r'-1)$ with *priority* at least 1, then, by Observation 4, there exists a good node p_g proposing v' in round r_g, where $r' - 3\mathcal{T} < r_g \leq r' - 1$ with $uCounter_g > 3\mathcal{T}$. Now with $r_g > (r' - 3\mathcal{T}) \geq (r_{lock} - 3\mathcal{T})$ (by definition of r'), we have $(r_{lock} - 3\mathcal{T}) < r_g \leq (r'-1)$, establishing (a).

Now, to establish (b), we show that $v_\ell(r_g - 1) = v'$. Since $uCounter_g > 0$, by line 17 of Sandglass, p_g collects only v' in round $(r_g - 1)$. Note that,

by Observation 2, all good nodes in round r_g, including p_g, collect at least one message proposing $v_\ell(r_g - 1)$ for round $(r_g - 1)$. Therefore, $v_\ell(r_g - 1)$ must be equal to v', *i.e.*, $v_\ell(r_g - 1) = v'$.

Having proved (a) and (b), we proved (*i*).

Proof of (*ii*) Consider the round r_g that exists by (*i*). We know that $(r_g - 1) \neq r_{lock}$. We now show that $(r_g - 1)$ can be neither smaller nor larger than r_{lock}, which leads to a contradiction.

Case 1 $(r_g - 1) < r_{lock}$

We are going to show that, under the assumption of $(r_g - 1) < r_{lock}$, it is possible to prove two statements, S1 and S2, that are in contradiction with each other.

S1: *There exists a round r, where* $r_g - 3\mathcal{T} < r \leq r_{\text{lock}} - 1$, *in which a good node proposes v with uCounter* $> 3\mathcal{T}$.

Since $v_\ell(r_g - 1) = v'$ and $v_\ell(r_{lock}) = v$, there must exist a round r_c between r_g and r_{lock} where v_ℓ changes from v' to v, *i.e.* $v_\ell(r_c - 1) = v'$ and $v_\ell(r_c) = v$.

By Observation 3, some good node in round r_c collects from round $(r_c - 1)$ some message proposing v with *priority* at least 1. Then, by Observation 4, there exists a good node p_v in round r_v, where $r_c - 3\mathcal{T} < r_v \leq r_c - 1$, proposing v with *uCounter* $> 3\mathcal{T}$.

Since $r_g \leq r_c \leq r_{lock}$, we have proved S1: there exists a round r_v, where $r_g - 3\mathcal{T} < r_v \leq r_{lock} - 1$, such that a good node p_v in r_v proposes v with *uCounter* $> 3\mathcal{T}$.

S2: *No good node in any round r, where* $r_g - 3\mathcal{T} < r \leq r_{\text{lock}} - 1$, *proposes v with uCounter* $> 3\mathcal{T}$.

Recall that, by (*i*), p_g proposes v' with $uCounter_g \geq 3\mathcal{T} + 1$ in round r_g. Consider any round r between $r_g - 3\mathcal{T}$ and r_g. By Corollary 4, there exists a good node in round r proposing v with *uCounter* at least $(uCounter_g - (r_g - r))$, which is a value greater than 0. Then, by Lemma 20, we can draw a first conclusion: no good node can propose v with $uCounter > 0$ for any round r, where $r_g - 3\mathcal{T} - 1 \leq r \leq r_g - 1$.

When r is equal to $(r_g - 1)$, this means that no good node proposes v with $uCounter > 0$ in round $(r_g - 1)$. Then, by Corollary 4, we can further infer that no good node proposes v with $uCounter > 3\mathcal{T}$ in any round between r_g and $(r_g + 3\mathcal{T})$. Since $r_g \geq (r_{lock} - 3\mathcal{T})$, *i.e.*, $(r_{lock} - 1) \leq r_g + 3\mathcal{T} - 1$, we can draw a second conclusion: for any round r, where $r_g \leq r \leq r_{lock} - 1$, no good node proposes v with $uCounter > 3\mathcal{T}$.

Combining our two conclusions, we have that no good node can propose v with $uCounter > 3\mathcal{T}$ in any round r, where $r_g - 3\mathcal{T} < r \leq r_{lock} - 1$, proving S2.

Since S1 and S2 contradict each other, and we were able to prove them under the assumption that $r_g - 1 < r_{lock}$, we conclude that Case 1 is impossible.

Case 2 $(r_g - 1) > r_{lock}$

Since $v_\ell(r_{lock}) = v$ and we proved that $v_\ell(r_g - 1) = v'$, then in some round r_c, where $r_{lock} < r_c \leq (r_g - 1)$, $v_\ell(r_c - 1) = v$ and $v_\ell(r_c) =$

v'. By Observation 3, some good node in round r_c must collect a message proposing v' with *priority* at least 1 from round $(r_c - 1)$.

Recall that r' is the earliest round in the range from r_{lock} to $(r_{start} + \overleftrightarrow{P_\ell} - 1)$, where some good node, currently in r', collects a message from round $(r' - 1)$ proposing $v' \neq v$ with *priority* at least 1; and that $r_g \leq r' - 1$. Therefore, $r_{lock} < r_c < r_g < r'$.

However, by assumption, r' is the earliest round in which some node collects a message proposing v' with *priority* at least 1. Contradiction.

This concludes the proof that F8 holds. Recall that, as we showed above, F8 implies F7:

For any r, where $r_{lock} < r \leq r_{start} + \overleftrightarrow{P_\ell} - 1$, $v_\ell(r) = v_\ell(r_{lock}) = v$, and all good nodes propose v for round r. (F7)

which is now also proved.

Step 2 Now, we are going to show that, for any good node p_g that is active at $T_D(r_{start})$, the *uCounter* of $v = v_\ell(r_{lock})$ at $T_D(r_{start})$ will be at least U^D. This is the condition upon which p_g will decide v.

The key technical hurdle we need to clear is to prove following fact:

A node that proposes $v' \neq v$ before $T_D(r_{start})$ can be at most in round $(r_{start} + \overleftrightarrow{P_\ell} - U^D - 1)$. (F9)

Assuming F9 holds, it follows easily that all good nodes that are active at $T_D(r_{start})$ must have decided by $T_D(r_{start})$. Here is why.

Since by F9 all nodes that propose in round $(r_{start} + \overleftrightarrow{P_\ell} - U^D)$ before $T_D(r_{start})$ must propose v, then, by line 17 of Sandglass, all nodes that propose in round $(r_{start} + \overleftrightarrow{P_\ell} - U^D + 1)$ before $T_D(r_{start})$ must propose v with *uCounter* at least 1. A simple inductive argument then shows that all nodes that ever propose in round $(r_{start} + \overleftrightarrow{P_\ell} - U^D + i)$ before $T_D(r_{start})$, where $1 \leq i < U^D$, propose v with *uCounter* at least i. With $i = U^D - 1$, messages sent for round $(r_{start} + \overleftrightarrow{P_\ell} - 1)$ before $T_D(r_{start})$ must propose v with *uCounter* at least $(U^D - 1)$. Note that by Lemma 10 and because p_g is active at step $T_D(r_{start})$, p_g enters round $(r_{start} + \overleftrightarrow{P_\ell})$ either at step $(T_D(r_{start}) - 1)$ or at step $T_D(r_{start})$. In both cases, p_g proposes v with *uCounter* at least U^D, *i.e.*, with *priority* at least $(6\mathcal{T} + 4)$, and decides by lines 21-22 of Sandglass. Therefore, if p_g is active at step $T_D(r_{start})$, it must have decided by step $T_D(r_{start})$.

To prove F9, we use again the notion of *bags* that we introduced in the proof for Agreement. We quickly review it below.

For each message m sent in round r, m's *bag* for round $(r - 1)$ is the set of messages collected by the sender of m in round $(r - 1)$.

Recall that, if some node p' sends a message $m^{v'}_{r'}$ proposing v' for round r', then there exists a chain of messages extending from round 1 to round r', where (a) each message on the chain proposes v', and (b) the i-th message on the chain was one of the highest *priority* messages collected from round i by the sender of the $(i + 1)$-th message.

To each message in the chain corresponds a bag: by definition, the bag of the chain's i-th message is the bag for round $(i - 1)$. Thus, in the chain there

exists exactly one bag per round, and at least one of the messages with the highest *priority* in each bag must be proposing v'.

Let $uCounter_i^{v'}$ be the value of *uCounter* of the i-th message on the chain. By line 17 of Sandglass, $\forall i : 2 \leq i < r'$: $uCounter_i^{v'} \geq uCounter_{i+1}^{v'} - 1$. Therefore, as we saw, the following holds:

$$\forall i : 2 \leq i < j \leq r' : uCounter_i^{v'} \geq uCounter_j^{v'} - (j - i). \tag{F3}$$

We are now ready to prove F9. We proceed by contradiction.

Assume there exists a node p' that before $T_D(r_{start})$ uses a message m' to propose v' in some round $r' > (r_{start} + \overleftrightarrow{P}_\ell - U^D - 1)$. Consider the chain of messages associated with m' and, in particular, the bags for every round from $(r_{lock} + 1)$ to $(r_{start} + \overleftrightarrow{P}_\ell - U^D - 1)$.

We will show that:

(i) Among the bags for rounds from $(r_{lock}+1)$ to $(r_{start}+\overleftrightarrow{P}_\ell-U^D-1)$, at most one in every U^1 bags can contain messages from good nodes. That is, among these $(r_{start} + \overleftrightarrow{P}_\ell - U^D - r_{lock} - 2)$ bags, $(r_{start} + \overleftrightarrow{P}_\ell - U^D - r_{lock} - 2 - \lceil \frac{r_{start}+\overleftrightarrow{P}_\ell-U^D-r_{lock}-2}{U^1} \rceil) = (\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2 - \lceil \frac{\overleftrightarrow{P}_\ell-6\mathcal{T}-U^D-2}{U^1} \rceil)$ of them contain only messages from defective nodes. We will call these *Def*-bags.

(ii) The number of messages sent by defective nodes for round $(r_{lock} + 1)$ to $(r_{start}+\overleftrightarrow{P}_\ell-U^D-1)$ before $T_D(r_{start})$ is not sufficient to fill all *Def*-bags.

Thus, it is impossible for a node that before T proposes $v' \neq v_d$ to advance up to round $(r_{start} + \overleftrightarrow{P}_\ell - U^D - 1)$, contradicting our assumption.

Proof of (i) We will show that among the bags for rounds $(r_{lock}+1)$ to $(r_{start}+\overleftrightarrow{P}_\ell - U^D - 1)$, at most one in every U^1 bags contains messages from good nodes.

By contradiction: assume there exist two bags containing messages from good nodes, for round r_1 and r_2 respectively, where $(r_{lock} + 1) \leq r_1 < r_2 \leq (r_{start} + \overleftrightarrow{P}_\ell - U^D - 1)$, and $r_2 - r_1 < U^1$. Consider now any message $m^g_{r_1}$ from a good node contained in the bag for round r_1. Since r_1 is within the lucky period that begins in r_{start}, by F7, $m^g_{r_1}$ proposes v. Similarly, any message $m^g_{r_2}$ from a good node contained in the bag for round r_2 proposes v.

Now, let us consider messages $m^{v'}_{r_1+1}$ and $m^{v'}_{r_2}$ on the chain. We showed above (F3) that $uCounter^{v'}_{r_1+1} \geq uCounter^{v'}_{r_2} - (r_2 - (r_1 + 1))$. Since $m^g_{r_1}$ proposing $v \neq v'$ is in the bag of round r_1, by line 17 of Sandglass, $uCounter^{v'}_{r_1+1} = 0$. Therefore, $uCounter^{v'}_{r_2} \leq uCounter^{v'}_{r_1+1} + (r_2 - (r_1 + 1)) = r_2 - r_1 - 1 < U^1$. Then, by line 20 of Sandglass, $priority^{v'}_{r_2} = 0$. However, recall that $m^{v'}_{r_2}$ is one of the messages with the largest *priority* among all messages in $m^{v'}_{r_2+1}$'s bag. Therefore, no message collected by $m^{v'}_{r_2+1}$ from round r_2 proposes v' with *priority* greater than 0. Note that $m^{v'}_{r_2+1}$ also collects $m^g_{r_2}$, which proposes v. Therefore, consider the set of values with the highest *priority* that $m^{v'}_{r_2+1}$ collected from round r_2. Either that set contains only v, when some v is proposed with *priority* greater than 0; or it contains both v and v', when both values are proposed with *priority* equal to 0. In either case, since $(r_2 + 1) \leq (r_{start} + \overleftrightarrow{P}_\ell - U^D) < (r_{start} + \overleftrightarrow{P}_\ell)$ is within the lucky period, and $v_\ell(r_2 + 1) = v_\ell(r_{lock}) = v$, $m^{v'}_{r_2+1}$ must propose v. However, by construction $m^{v'}_{r_2+1}$ should propose v'. Contradiction.

Proof of (*ii*) We have just established that, among the bags for rounds from $(r_{lock} + 1)$ to $(r_{start} + \overleftrightarrow{P}_\ell - U^D - 1)$, $(r_{start} + \overleftrightarrow{P}_\ell - U^D - r_{lock} -$

$2 - \lceil \frac{r_{start} + \overleftrightarrow{P}_\ell - U^D - r_{lock} - 2}{U^1} \rceil$) *Def*-bags contain only messages from defective nodes.

Since each bag contains at least $\mathcal{T}$ messages, *Def*-bags contain at least $\mathcal{T} \cdot (\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2 - \lceil \frac{\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2}{U^1} \rceil)$ messages.

Let us now compute the largest number of messages these *Def*-bags can contain.

Def-bags can only contain messages sent for round $(r_{lock} + 1)$ to $(r_{start} + \overleftrightarrow{P}_\ell - U^D - 1)$ by defective nodes. By Lemma 12, $T^{\forall}_{r_{lock}}$ is the earliest step where some defective node can be in round $(r_{lock} + 1)$. Then, the messages in *Def*-bags must have been sent from step $T^{\forall}_{r_{lock}}$ to step $(T_D(r_{start}) - 1)$. By Lemma 17, the number of messages sent by defective nodes in the time interval from $T^{\forall}_r$ to $(T^{\forall}_{r+1} - 1)$ is at most $(\mathcal{T} - 1)$. Since $T_D(r_{start}) = T^{\forall}_{r_{start} + \overleftrightarrow{P}_\ell}$, the period from step $T^{\forall}_{r_{lock}}$ to step $(T_D(r_{start}) - 1)$ covers $(r_{start} + \overleftrightarrow{P}_\ell - r_{lock}) = (\overleftrightarrow{P}_\ell - 6\mathcal{T})$ such intervals; thus, the number of messages sent by defective nodes in this period is at most $(\mathcal{T} - 1) \cdot (\overleftrightarrow{P}_\ell - 6\mathcal{T})$.

Recall that *Def*-bags contain at least $\mathcal{T} \cdot (\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2 - \lceil \frac{\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2}{U^1} \rceil)$

messages. Therefore, we have

$$(\mathcal{T}-1)\cdot(\overleftrightarrow{P}_\ell - 6\mathcal{T}) \geq \mathcal{T}\cdot(\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2 - \lceil\frac{\overleftrightarrow{P}_\ell - 6\mathcal{T} - U^D - 2}{U^1}\rceil)$$

$$\Rightarrow(\mathcal{T}-1)\cdot C \geq \mathcal{T}\cdot(C - U^D - 2 - \lceil\frac{C - U^D - 2}{U^1}\rceil)$$

$$\text{(where C} = \lceil\tfrac{(6\mathcal{T}-1)\cdot U^D + 18\mathcal{T}}{5}\rceil\text{, and } \overleftrightarrow{P}_\ell = 6\mathcal{T} + C)$$

$$\Rightarrow U^D + 2 + \lceil\frac{C - U^D - 2}{U^1}\rceil \geq \frac{C}{\mathcal{T}}$$

$$\Rightarrow U^D + 3 + \frac{C - U^D - 2}{U^1} > \frac{C}{\mathcal{T}} \qquad \text{(Since } (\tfrac{C-U^D-2}{U^1} + 1) > \lceil\tfrac{C-U^D-2}{U^1}\rceil)$$

$$\Rightarrow U^D + 3 > \frac{5C + U^D + 2}{6\mathcal{T}} \qquad \text{(Since } U^1 = 6\mathcal{T})$$

$$\Rightarrow C < \frac{6\mathcal{T}(U^D+3) - U^D - 2}{5} = \frac{(6\mathcal{T}-1)U^D + 18\mathcal{T} - 2}{5}$$

However, since $C = \lceil\frac{(6\mathcal{T}-1)\cdot U^D + 18\mathcal{T}}{5}\rceil > \frac{(6\mathcal{T}-1)U^D + 18\mathcal{T} - 2}{5}$, we have a contradiction. □

Lemma 22 (Termination with probability 1)**.** *Every good node that remains active decides with probability 1.*

Proof. By Observation 5, for any round r_{start}, if r_{start} is a lucky round, then all good nodes that are active in step $T_D(r_{start})$ decide a value in round $r_{start} + 6\mathcal{T} + \lceil\frac{(6\mathcal{T}-1)\cdot U^D + 18\mathcal{T}}{5}\rceil$. Let $\overleftrightarrow{P}_\ell = 6\mathcal{T} + \lceil\frac{(6\mathcal{T}-1)\cdot U^D + 18\mathcal{T}}{5}\rceil$.

Let $S = \{1 + k\cdot(\overleftrightarrow{P}_\ell + 1) | k \in N\}$. S is a set containing infinitely many numbers of rounds, that the events of each of them being lucky are mutually independent.

Let $T^\forall(r)$ be the earliest step where all good nodes are in round r. For any round r, $T_1(r)$, which is earliest step that any good node can be in round r, is no earlier than $T^\forall(r-1)$ by Lemma 12; and $T_D(r)$ is defined as $T^\forall(r + \overleftrightarrow{P}_\ell)$. Consider the period consisting of the steps from $T_1(r)$ to $T_D(r) - 1$, i.e., from $T^\forall(r-1)$ to $T^\forall(r + \overleftrightarrow{P}_\ell) - 1$; this period covers all rounds from r to $(r + \overleftrightarrow{P}_\ell - 1)$. For this

period to be lucky and for r to be a lucky round, we require any node that must randomly select the value it will propose in any round between r and $r + \overleftrightarrow{P}_\ell - 1$ to select the lucky value v_ℓ for its current round.

Consider the events that correspond to rounds in S being lucky. Since the lucky periods for rounds in S are not overlapping, these events are mutually independent.

Now we are going to show that all rounds in S are lucky with non-zero probability. Consider round r in S. In each step of r's lucky period, there are at most N nodes in the system. Each node that makes a random choice in one of the rounds covered by the lucky period chooses the round's lucky value with probability $\frac{1}{2}$. Therefore, in every step of the lucky period, the probability that all nodes that make random choices select the lucky value for their current round is at least $\frac{1}{2^N}$. By Lemma 11, it takes at most $\mathcal{T} \cdot (\overleftrightarrow{P}_\ell + 1)$ steps from $T_1(r)$ to $(T_D(r) - 1)$. Therefore, the probability that any round in S is lucky is at least $\frac{1}{2^{N \cdot \mathcal{T} \cdot (\overleftrightarrow{P}_{\ell+1})}} > 0$.

Now, consider any good node p_g that joins in round r_g at any step T and stays active. Recall that, by Lemma 11, good nodes are guaranteed to eventually reach any arbitrary round. Since there are infinitely many rounds r in S where $T_D(r) > T$, with probability 1 there exists a round $r \in S$ such that (1) r is a lucky round; and (2) $T_D(r) \geq T$. Then, by Lemma 11, p_g will eventually reach $T_D(r)$ and, by Observation 5, decide. □

APPENDIX B

GORILLA CORRECTNESS

B.1 Sandglass Plus

In this section, we first introduce the SM+ model, which is nearly identical to that of Sandglass (§3), with the exception that it permits defective nodes to receive messages sent by other defective nodes within the same step, subject to certain constraints.

We then prove that Sandglass remains correct in this new model by satisfying Validity, Agreement, and termination with probability 1. Fortunately, the correctness proof of Sandglass requires only minor modifications to the proofs of two lemmas to be applicable to the SM+ model. We will show in Section B.1.2 the two lemmas that require change, and refer the readers to the Sandglass paper for the rest of the proof.

B.1.1 The SM+ Model

SM+ and the Sandglass model (SM) largely make the same set of assumptions. We show the only difference here and refer the readers to the rest of the model in the Sandglass (§3.1): SM assumes that *if in step t a node p_i receives message m with $Receive_i$, then m was sent in some step $t' < t$*. In SM+, we weaken this assumption, by allowing defective nodes to non-recursively receive messages sent from defective nodes within the same step. Formally, if in step t a node p_i receives message m from p_j with $Receive_i$, then m was sent in some step $t' < t$ when at

least one of p_i and p_j is good, or $t' \leq t$ when both p_i and p_j are defective and m does not contain in its coffer a message that is also sent in s.

B.1.2 Sandglass is Correct in SM+

The proof of correctness for Sandglass in SM+ closely resembles the proof for SM, with the exception of Lemma 12 and Lemma 18 in the Sandglass proof (§A). While the statements of these lemmas remain unchanged, their proofs require minor modifications.

It is perhaps surprising that so little of the proof needs changing when moving from SM to SM+. The reason is that the original Sandglass proof assumes that all the messages generated by defective nodes contribute to their progress, regardless of when they are actually received. Specifically, when estimating the maximum possible round that defective nodes can be in, the proof considers the totality of messages generated by defective nodes during the execution and divides it by the size of the threshold of messages that must be received to advance to a new round. This best-case scenario for defective nodes already accounts for the additional flexibility that SM+ affords to defective nodes. In particular, the original proof already accounts for the possibility, allowed in SM+, that defective nodes receive, in a given step, messages that defective nodes sent in that same step–even though SM disallows such executions.

Lemma 12 in Sandglass. *At any step T, any defective node is at most one round ahead of any good node.*

Proof. By contradiction. Assume that there exists an earliest step, T, where some

defective node p is more than one round ahead of a good node p_g, *i.e.*, at T node p is in some round r and node p_g is in round $r_{p_g} \leq (r-2)$.

Note that no good node is in round $(r-1)$ or larger before T; otherwise, by Lemma 3 in Sandglass (§A) , all good nodes would be in round $(r-1)$ or larger at T, contradicting $r_{p_g} \leq (r-2)$. Therefore, defective node p received no messages from good nodes for round $(r-1)$ by T.

Consider the earliest step $T' \leq T$ where some defective node is in round $(r-1)$. Since T is the first step where some defective node is more than a round ahead of a good node, all good nodes must be in round $(r-2)$ or larger at T'; but, as we just showed, no good node is in round $(r-1)$ or larger before T. Therefore, all good nodes must be in round $(r-2)$ from T' until T.

Consider the k consecutive steps from T' to T. Let the number of messages generated by good nodes and defective nodes in each step be, respectively, $g_1, ..., g_k$ and $d_1, ..., d_k$. Since up to and including step T node p has received for round $(r-1)$ only messages from defective nodes, and yet p is in round r at T, by line 6 of Sandglass, $\Sigma_{i=1}^{i=k} d_i - 1 \geq \mathcal{T}$ and thus, by Lemma 1 in Sandglass (§A) , $\Sigma_{i=2}^{i=k-1} g_i \geq \mathcal{T}$. Since by assumption every step includes at least one good node (*i.e.*, $g_1 > 0$), we have that $\Sigma_{i=1}^{i=k-1} g_i > \mathcal{T}$. Recall that during these $(k-1)$ steps all good nodes are in round $(r-2)$; then, all messages $g_1, ..., g_{k-1}$ are for round $(r-2)$ and will all be received by all good nodes by T. By line 6 and line 7, then, *all* good nodes (including p_g) must be in round $(r-1)$ at T. This contradicts our assumption and completes the proof. □

Lemma 18 in Sandglass. *Suppose a good node p_g is in round r at step T, and a node p_d is in round r_d at step $T' \leq T$. If p_d does not collect any messages from good nodes in any round $(r-i)$, where $0 \leq i < k\mathcal{T}$, then $r_d \leq (r-(k-1))$.*

Proof. To prove the corollary, we compute the maximum number of messages D_{max} that a defective node p_d can collect during the $k\mathcal{T}$ rounds when it does not collect any message from good nodes. To help us count these messages, for any $1 \leq i \leq k\mathcal{T}$, denote by $T_{(r-k\mathcal{T}+i)}$ the earliest step for which all good nodes are at least in round $(r - k\mathcal{T} + i)$.

Recall that, to be collected by p_d at step T', a message from a good node must have been generated no later than step $(T' - 1) \leq (T - 1)$, and a message from a defective node must have been generated no later than step $T' \leq T$. Then, we partition the execution of the system up to step T into three time intervals, and compute, for each interval, the maximum number of messages generated during these intervals that p_d could have collected for rounds $(r - k\mathcal{T} + 2)$ or larger.

I1: Up to step $(T_{(r-k\mathcal{T}+1)} - 1)$.

By definition of $T_{(r-k\mathcal{T}+1)}$, some good node is in some round $r' < r - k\mathcal{T} + 1$ at step $(T_{(r-k\mathcal{T}+1)} - 1)$. Therefore, neither a defective node nor a good node can be in some round $r'' > r - k\mathcal{T} + 1$ at step $(T_{(r-k\mathcal{T}+1)} - 1)$, respectively because of Lemma 5 and Corollary 2 in Sandglass (§A) . Therefore, during this interval, no messages were generated for rounds $(r - k\mathcal{T} + 2)$ or larger.

I2: From $T_{(r-k\mathcal{T}+1)}$ up to $(T_r - 1)$.

By assumption, p_d only collects messages generated by defective nodes throughout interval I2. We further split I2 into i consecutive subintervals, each going from $T_{(r-k\mathcal{T}+i)}$ up to $(T_{(r-k\mathcal{T}+i+1)} - 1)$ for $1 \leq i \leq (k\mathcal{T} - 1)$. By Lemma 10 in Sandglass (§A) , in each of these sub-intervals defective nodes can generate at most $(\mathcal{T} - 1)$ messages. Therefore, the number of messages generated by defective nodes during I2 is at most $(\mathcal{T} - 1) \cdot (k\mathcal{T} - 1)$.

I3: From T_r to T.

Once again, by assumption, p_d only collects messages generated by defective nodes throughout interval I3. There are two cases:

- T precedes T_r.

  If so, defective nodes trivially generate no messages during I3.

- T does not precede T_r.

  By assumption, some good node p_g is in round r at T, where it collects all messages generated by good nodes before T; further, since p_g is still in round r, the messages for round r sent by good nodes before T must be fewer than $\mathcal{T}$. Finally, since p_g is in round r at T, by Lemma 3 in Sandglass (§A) , in all steps preceding T no good node can be in round $(r+1)$ or higher. We then conclude that from step T_r and up to $(T-1)$ good nodes generated at most $(\mathcal{T}-1)$ messages, all for round r. Let the number of messages generated by good nodes and defective nodes starting from T_r to T be, respectively, $g_1, ..., g_k$ and $d_1, ..., d_k$. Then we have $\Sigma_{i=1}^{k-1} g_i < \mathcal{T}$. By Lemma 1 in Sandglass (§A) , we then have $\Sigma_{i=1}^{k} d_i < \mathcal{T}$, *i.e.*, during I3 defective nodes generate fewer than $(\mathcal{T}-1)$ messages.

Adding the messages generated in the three intervals, we find that D_{max}, the maximum number of messages that p_d could have collected up to and including step T for rounds $(r-k\mathcal{T}+2)$ or larger, is smaller than $(\mathcal{T}-1)\cdot k\mathcal{T}$; at the same time, since by assumption p_d is in round r_d, D_{max} must equal at least $(r_d-(r-k\mathcal{T}+2))\cdot\mathcal{T}$. Therefore, we have that $(r_d-(r-k\mathcal{T}+2))\cdot\mathcal{T} < (\mathcal{T}-1)\cdot k\mathcal{T}$, which implies $r_d \leq r-(k-1)$, proving the corollary.

□

Finally, we have our intended theorem.

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

Proof for the theorem directly follows from Lemma 2, 14, 15 in Sandglass (§A).

To prove the correctness of Gorilla, we first show a mapping in two steps from a Gorilla execution to an execution of Sandglass in SM+ (§B.2). Then, given that Sandglass is correct in SM+, we leverage this mapping to proof safety (§B.3) and liveness (§B.4) for Gorilla.

B.2 Scaffolding

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 4.3.1, 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's *peek* query, Byzantine nodes active in any step s can learn the

result of a VDF computed by Byzantine nodes finishing in step s even 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, 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 7. *Consider an execution η_G in GM and an execution η_G^+ in GM+. We say η_G^+ is a reorg of η_G 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 2. *There exists a mapping* REORG *that maps an execution η_G in GM to an execution η_G^+ in GM+, denoted $\eta_G^+ = \text{REORG}(\eta_G)$, such that η_G^+ is a reorg of η_G.*

Proof. Consider any Gorilla execution η_G. We are going to construct an execution η_G^+ in GM+ that satisfies REORG-1,2,3.

First, we specify how correct and Byzantine nodes join and leave in η_G^+. For each correct node p in η_G, a corresponding correct node p^+ in η_G^+ joins and leaves the system at the same steps as p in η_G. Consider any step s in η_G^+, and let c be the number of correct nodes in step s. We make $(c-1)$ Byzantine nodes join at the beginning of step s and leave at the end of step s.

Let the set of valid messages sent by Byzantine nodes in η_G be $\mathcal{M}_B$. Note that this set of valid messages sent by Byzantine nodes can be larger than the set of valid messages correct nodes received from Byzantine nodes due to Byzantine omissions.

Our proof proceeds in two steps. We overview them and then explain them in detail:

Step 1 For any $m \in \mathcal{M}_B$, we assign a unique *shell*, (s, b), identified by a step s and a Byzantine node b in η_G^+, for the K Get() calls of VDF calculation for m. Note that any node can only make one Get() call in a tick, and it takes K Get() calls to get vdf_m.

We prove four claims about the shells, which are useful later to prove the same messages can be generated in η_G and in η_G^+.

Step 2 We prove by induction that correct nodes will receive and send the same messages in η_G^+ as in η_G, and the same valid messages are sent by Byzantine nodes at the same step or earlier. Then, it immediately follows that REORG-1,2,3 are satisfied.

Step 1

We run Algorithm 5 to assign shells for the VDF calculation of messages in $\mathcal{M}_B$. The algorithm operates as follows: we maintain two variables within a loop, s and CandidateVDF, where s denotes the step number, initially set to -1, and CandidateVDF represents a set of VDFs, starting as an empty set. During each loop iteration, s is incremented by 1. The algorithm adds VDFs whose first units are calculated in step s of η_G to the CandidateVDF set. While there exists an available shell in step s, the algorithm assigns this shell to a VDF, vdf, from CandidateVDF. The selected vdf's last unit should be calculated the earliest in η_G and the algorithm then removes it from CandidateVDF. This process continues until no free shells remain in step s. Subsequently, the algorithm moves to the next iteration of the loop and repeats these steps.

Now we prove the following claims are true about the assignment:

Claim 3. *For any $m \in \mathcal{M}_B$, if the first Get() call for vdf_m is in step i, then the shell*

Algorithm 5 Algorithm for reorganizing VDF units

```
1:  procedure REORG
2:      s ← −1
3:      CandidateVDFs← ∅
4:      loop
5:          s ← s + 1; B_s ← the set of Byzantine nodes at step s
6:          CandidateVDFs ← CandidateVDFs ∪ {vdf — vdf's first unit is calculated in step s in η_G}
7:          while there's a free shell (s, p ∈ B_s) do
8:              vdf← a VDF result in CandidateVDFs whose last unit is calculated the earliest in η_G
9:              Assign (s, p) to vdf
10:             CandidateVDFs ← CandidateVDFs\{vdf}
```

assigned to vdf_m in η_G^+ is in step i or later.

Proof. Since vdf_m is not added to the *CandidateVDFs* set at line 6 of Algorithm 5 until s is increased to i (in line 5), the step of the shell that vdf_m can be assigned to is at least i (line 9). □

Before stepping into Claim 4, we show a useful observation following from Algorithm 5.

Observation 6. *Consider any step s in η_G^+. We note two possible scenarios of shells at step s, such that if either of these scenarios happens, the calculation of VDFs assigned to shells later than s in η_G^+ must start in a step later than s in η_G:*

A free shell exists at step s in η_G^+. *When the loop for s finishes, if there is a free shell at (s, p), then the CandidateVDFs set is empty at the end of the iteration for s. That is, any vdf assigned to a shell in a later step starts its calculation at a step $s' > s$ in η_G.*

A shell at step s is assigned to a *vdf* that is not the earliest to finish in η_G^+ *, among all the vdfs that are not assigned to a shell yet. Consider the scenario in which a vdf is assigned to a shell at s at line 9 (it is the earliest to finish among all VDFs in CandidateVDFs), but it is not the earliest to finish among all the remaining VDFs. Then the calculation of the remaining VDFs, including those VDFs whose calculation is finished earlier than vdf, must start later than s in η_G, because they are not in the CandidateVDFs set yet.*

We are ready to prove Claim 4.

Claim 4. *For any $m \in \mathcal{M}_B$, if the last Get() call for vdf_m is in step i, then the shell assigned to vdf_m in η_G^+ is in step i or earlier.*

Proof. We prove this by contradiction. Consider the first step i in η_G, such that the last Get() call of a VDF, vdf^*, is in step i, but vdf^* is not assigned to a shell in η_G^+ by the end of i-th iteration of the loop.

Consider the largest step $j \leq i$ such that, one of the scenarios in Observation 6 happens. If no such j exists, we take j to be -1. By Observation 6, for all the VDFs that are already assigned to the shells in steps $[j+1, i]$, their first Get() calls are after step j. Furthermore, since they are already assigned to shells, their last unit is no later than vdf^*'s, *i.e.*, their last Get() calls are in or before step i . We call this set of vdfs, VDF_{occupy}. Then, we have all of the Get() calls of VDF_{occupy} are in steps $[j+1, i]$ in η_G.

Note that there are no free shells in $[j+1, i]$ (otherwise, j would be larger). Let the number of Byzantine nodes in any step s be b_s. Then, the size of VDF_{occupy} is $\Sigma_{s=j+1}^{i} b_s$. Therefore the number of Get() calls for VDF_{occupy} in steps $[j+1, i]$ is $K \cdot \Sigma_{s=j+1}^{i} b_s$ in η_G.

Note that in η_G, the number of Byzantine nodes at any step s is at most b_s. Therefore, the total number of Get() calls that can be made in steps $[j+1, i]$ in η_G is at most $K \cdot \Sigma_{s=j+1}^{i} b_s$, and one of them is the last Get() call of vdf^*. Therefore, there are not enough ticks available to make all Get() calls for VDF_{occupy} in η_G. A contradiction. □

Claim 5. *Consider any two VDFs, vdf_1 and vdf_2, reserved respectively at steps s_1 and s_2 in η_G^+. If the last Get() call of vdf_1 is before the first Get() call of vdf_2 in η_G, then $s_1 \leq s_2$.*

Proof. By line 6, vdf_2 must be added to the CandidateVDFs set after vdf_1. By line 8, vdf_1 must be assigned to its shell before vdf_2. Note that, by line 5, shells are assigned in non-decreasing step order; therefore, $s_1 \leq s_2$. □

Claim 6. *Consider any three vdfs, vdf_1, vdf_2, and vdf_3, reserved respectively at steps s_1, s_2 and s_3 in η_G^+. If, in η_G, the last Get() call of vdf_1 is before the first Get() call of vdf_2, and the last Get() call of vdf_2 is before the first Get() call of vdf_3, then $s_1 < s_3$.*

Proof. Let the tick of the first Get() call of vdf_2, and vdf_3 in η_G be t_2^f, and t_3^f.

Note that the last Get() call of vdf_1 is before t_2^f in η_G. By Claim 4, vdf_1 must be assigned to step $\lfloor (t_2^f - 1)/K \rfloor$ or earlier. By Claim 3, vdf_3 must be assigned to $\lfloor t_3^f/K \rfloor$ or later. Since $t_2^f \leq t_3^f - K$, *i.e.*, $(t_2^f - 1) < t_3^f - K$, we have $s_1 \leq \lfloor (t_2^f - 1)/K \rfloor < \lfloor t_3^f/K \rfloor \leq s_3$, *i.e.*, $s_1 < s_3$.

□

Now in η_G^+, all the Byzantine nodes join and leave at the boundaries and stay for a single step; and for each valid message m sent by a Byzantine node in η_G, we have assigned a unique shell (s, b) for some s and b^* to it. Then, if b can

receive the messages contained in m's message coffer, M_m, then b will make K Get() calls in step s for input $(M_m, nonce_m)$, and therefore b will be able to send m in η_G^+.

We construct η_G^+ so that if a Byzantine node is able to send a (valid) message m, it sends m to all Byzantine nodes in the next step. Every Byzantine node forwards all the messages it has received in a step to all the Byzantine nodes in the next step. Furthermore, if m is received at tick t' by a correct node c in η_G, one Byzantine node who has m at tick $(t'-1)$ will send m to c at tick $(t'-1)$ in η_G^+.

We will show in Step 2 that for each valid message m sent by a Byzantine node in η_G and the shell (s, b) assigned to it, b can indeed receive the messages contained in m's message coffer, and can therefore send m at s. Furthermore, any correct node c^+ at step s in η_G^+ can receive the same set of messages as its corresponding node c in η_G at step s, and therefore send the same message at s.

Step 2

We will prove by induction that we can construct η_G^+ such that any message received and sent in η_G by correct nodes is received and sent in η_G^+ at the same step, and any valid message sent in η_G by a Byzantine node is sent in η_G^+ at the same or earlier step by a Byzantine node and is received at the same step as in η_G by correct nodes.

Recall that in GM+, Byzantine nodes can peek VDF results of other Byzantine nodes that complete at the same step.

Base case In step 0, we show it is possible to (i) make all correct nodes in η_G^+

send the same messages sent by their corresponding correct nodes in η_G at step 0, and (*ii*) for any shell $(0, p_{sh})$ that is reserved for message m_{sh}, make p_{sh} send m_{sh}.

First, consider the valid messages that are sent by correct nodes in step 0 in η_G. In step 0, there were not enough ticks to generate a VDF. Therefore, all correct nodes receive no (valid) message in η_G, *i.e.*, the message coffer of any valid message sent by a correct node in step 0 in η_G is empty. Therefore, by making all the correct nodes have the same initial values as in η_G and pick the same nonces, any (valid) message sent in η_G can also be sent in η_G^+ in step 0.

Second, consider any message m_{sh} whose shell is in step 0 in η_G^+. Consider the message coffer M_{sh} of m_{sh}. Note that in η_G, for any $m \in M_{sh}$, m's last Get() call must be before m_{sh}'s first Get() call. Therefore, by Claim 5, m's shell must also be in step 0. Since step 0 is the earliest step, by Claim 6, m does not contain any messages in its message coffer, otherwise messages in m's message coffer would have been assigned to a step earlier than step 0. Then, p^+ can include the messages in M_{sh} in its message coffer in η_G^+ by peeking the vdf_m, therefore, p^+ can send m_{sh} in step 0. Again, by picking the same nonce, m_{sh} can be sent in η_G^+ in step 0.

We make the nodes send these messages in η_G^+ in the following way:

- Correct nodes send their messages to all the nodes.
- Byzantine nodes send their messages to all the Byzantine nodes.
- Consider a correct node p_c that receives a message m sent by a Byzantine node in η_G at step 1. Note that by Claim 4, m must be assigned to a shell in step 0. Let that shell be $(0, b_m)$. b_m sends m to p_c at step 0

in η_G^+.

Induction hypothesis Up until step k, any message sent in η_G by a correct node is sent in η_G^+ at the same step. Byzantine nodes can send all the messages whose shells are at step K. Messages received by correct nodes up until step $(k+1)$ are the same as in η_G. Byzantine nodes receive all the messages from correct nodes and Byzantine nodes.

Induction step Consider messages sent at step $(k+1)$ in η_G.

First, we prove correct nodes can send the same messages in step $(k+1)$ in η_G^+ as in η_G. By the induction hypothesis and due to synchrony, correct nodes receive the same set of messages from Byzantine nodes and correct nodes in step $(k+1)$ in η_G^+ and in η_G. By making all the correct nodes select the same nonces in η_G^+ as in η_G, correct nodes are going to send the same messages in η_G^+ as in η_G.

Second, we prove that Byzantine nodes can send all the messages whose shells are at step $(k+1)$ in η_G^+. Consider any message m_{sh} whose shell is at step $(k+1)$ and any m_M in m_{sh}'s message coffer in η_G. There are two possibilities for m_M.

m_M is sent by a Byzantine node Then the last Get() call for the VDF of m_M is before the first Get() call for the VDF of m_{sh} in η_G. Then, by Claim 5, the shell reserved for m_M is in a step s_M, where $s_M \leq (k+1)$. Note that in η_G^+, Byzantine nodes in a step can peek at the VDF results for messages from Byzantine nodes that finish within the same step; and, by Claim 6, m_M does not contain any message whose shell is also in step $(k+1)$ or later. It follows that m_M can be included in the message coffer of m_{sh} in η_G^+.

m_M is sent by a correct node By Claim 3, m_{sh} is calculated no later than step $(k+1)$ in η_G. Therefore, m_M must be sent in a step no later than step k in η_G. By the induction hypothesis, m_M is sent in η_G^+. Therefore, m_M can be included in m_{sh}'s coffer in η_G^+.

Therefore, by picking the same nonces, Byzantine nodes can also send the same messages as in η_G at step $(k+1)$ in η_G^+ whose shells are at step $(k+1)$.

We make the nodes send messages in η_G^+ in the following way:

- Correct nodes send their messages to all the nodes.
- Byzantine nodes send their messages to all the Byzantine nodes.
- Byzantine nodes forward messages they received to all the Byzantine nodes.
- Consider a correct node p_g that receives a message m sent by a Byzantine node in η_G at step $(k+2)$. By Claim 4, m must be assigned to a shell that is no later than $(k+1)$. We just showed above that Byzantine nodes can send all the messages whose shells are at step $(k+1)$. Combining with induction hypothesis, some Byzantine node b_m must have received or generated m. We make b_m sends m to p_g at step $(k+1)$ in η_G^+.

Now we have proven that correct nodes receive and send the same messages in η_G as in η_G^+. Note that we also proved that Byzantine nodes can send all the messages at the step where their shells are. By Claim 4, it directly follows that for any valid message m sent by a Byzantine node in η_G, a Byzantine node sends the same messages in the same step or earlier in η_G^+ (REORG-3).

In summary, we have constructed η_G^+ in GM+ that satisfies REORG-1,2,3.

□

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+, a model 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: But 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* (§B.1.2). 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. We map messages by simply translating the data structure:

Definition 8. *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 $\hat{m} =$ 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* MAPM$(\mathcal{M})$.

Thus, we can define the execution mapping:

Definition 9. *Consider an execution η_G^+ in GM+ and an execution η_S^+ in SM+. We say η_S^+ is an interpretation of η_G^+ 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. *Node $\hat{p}$ 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. *Node $\hat{p}$ receives $\hat{m}$ at step s in η_S^+ iff p receives m at step s in η_G^+.*

Lemma 3. *Consider any execution η_G in GM, and an execution η_G^+ in GM+ is a reorg of η_G. There exists a mapping* INTERPRET *that maps η_G^+ to an execution η_S^+ in SM+, denoted as $\eta_S^+ =$ INTERPRET(η_G^+), such that η_S^+ is an interpretation of η_G^+.*

Proof. For execution η_G^+, we construct the interpretation of η_G^+, η_S^+, in SM+. First, for every p in η_G^+, we add a corresponding $\hat{p}$ to η_S^+ such that:

- $\hat{p}$ joins and leaves at the same steps that p joins and leaves, respectively.
- $\hat{p}$ has the same initial value as p.
- If p is a Byzantine node, then $\hat{p}$ is a defective node; otherwise, $\hat{p}$ is a good node.

The number of Byzantine nodes in a step of η_G^+ is smaller than the number of correct nodes at each step and the number of defective and good nodes are equal to those of Byzantine and correct nodes, respectively; therefore, the number of defective nodes in η_S^+ is fewer than that of good nodes at each step. Thus, Condition 1, 2 and 3 are satisfied.

We now construct the messages sent in η_S^+ such that η_S^+ is an interpretation of η_G^+. Specifically, we require Condition 4 and Condition 5 in Definition 9 to be satisfied for the messages sent. We prove this by induction on steps. Note that messages are constructed inductively alongside the induction.

Induction Base Consider any node p in η_G^+ at the first step, and message m that it generates at the first step. We prove the claim holds for the first step, conditioned on whether p is a correct node or not.

p **is a correct node** Note that p and $\hat{p}$ have the same initial value v_p by construction. We now prove that $\hat{p}$ will send $\hat{m} = \text{MAPM}(m)$ to all nodes at the first step in η_S^+.

Since there are not enough ticks to generate a VDF, p does not receive any message in the first step, therefore, m does not contain any message in its message coffer. Therefore, $m = (p, r = 1, v = v_p, priority = 0, uCounter = 0, M = \emptyset, \cdot, \cdot)$. Note that $\hat{p}$ cannot receive any message in the first step, either. The message $\hat{p}$ sends at the first step, by Sandglass, is $(\hat{p}, uid = 1, r = 1, v = v_p, priority = 0, uCounter = 0, M = \emptyset)$, which is equal to $\hat{m}$. Note that in η_S^+, since good nodes are synchronously connected, all the good nodes in the second step will receive $\hat{m}$. Note that it's possible for a defective node not to receive $\hat{m}$, by performing an omission failure, or to receive $\hat{m}$ in any step, being asynchronously connected to other nodes.

p **is a Byzantine node** Note that in SM+, defective nodes at step s can receive any message m sent at s, as long as m does not contain in its coffer a message that is also sent at s. Again, we will prove $\hat{m}$ can and will be sent at the first step in η_S^+.

We first consider any message m generated by a Byzantine node p, whose message coffer is empty. With the same argument for a correct node p in the first step of η_G^+, $\hat{p}$ will send $\hat{m}$ in the first step of η_S^+. Again, note that it's possible for any node not to receive $\hat{m}$, when $\hat{p}$ performs an omission failure, or to receive $\hat{m}$ in any step, since $\hat{p}$ is

asynchronously connected to other nodes.

Second, let's consider a message m generated by a Byzantine node p, whose message coffer is not empty. Note that the number of messages that can be included in m's coffer is at most the number of Byzantine nodes in the first step, which is smaller than $\mathcal{T}$. We have $m = (p, r = 1, v, priority = 0, uCounter = 0, M, \cdot, \cdot)$. Consider any message $m_{pk} \in M$, m_{pk} must be sent by a Byzantine node and the *vdf* in m_{pk} is a Peek() result. By specification of Peek(), m_{pk} must be generated in the first step, and not contain any message in its message coffer. We have proven above that $\hat{m}_{pk}$ will be sent in the first step of η_S^+. Note that message coffer of $\hat{m}_{pk}$ is also empty. Therefore, we make the scheduler deliver $\hat{m}_{pk}$ to the defective node $\hat{p}$ and therefore $\hat{p}$ will include $\hat{m}_{pk}$ in its message coffer. In summary, for every $m_{pk} \in M$, $\hat{p}$ will receive $\hat{m}_{pk}$, and $|M| < \mathcal{T}$. Therefore, $\hat{p}$ will send $(p, r = 1, v, priority = 0, uCounter = 0, \text{MAPM}(M))$, which is equal to $\hat{m}$.

Induction Hypothesis Node $\hat{p}$ receives $\hat{m}$ at step $s' \leq s$ in η_S^+, iff p receives m at step s' in η_G^+ (Condition 4). Node $\hat{p}$ sends $\hat{m}$ at step $s' \leq s$ in η_S^+, iff p generates a message m in η_G^+ at step s' (Condition 5).

Induction Step Now we prove that the claim holds for step $s + 1$, conditioned on whether p is a correct node or not.

p is a correct node We prove Condition 4 and Condition 5 separately.

Condition 4 First, we will prove that if p receives m at step $(s + 1)$ in η_G^+, then $\hat{p}$ receives $\hat{m}$ at step $(s + 1)$ in η_S^+.

If m is generated by a correct node p_c, m must be sent at step s in η_G^+. By the induction hypothesis, $\hat{m}$ must be sent by a good

node $\hat{p}_c$ at step s in η_S^+. Note that in SM+, the network between good nodes is synchronous. Therefore, $\hat{p}$ receives $\hat{m}$ at $(s+1)$ in η_S^+. If m is generated by a Byzantine node p, m could have been generated at any step $s' \leq s$ and finally sent to p at s in η_G^+. By the induction hypothesis, $\hat{m}$ must be sent by a defective node $\hat{p}$ at step s' in η_S^+. Note that in SM+, the network between good nodes and defective nodes is asynchronous. Therefore, we can make the scheduler deliver $\hat{m}$ to $\hat{p}$ at $(s+1)$ in η_S^+.

Condition 5 Now we prove that if p generates a message m in η_G^+ at step $(s+1)$, then $\hat{p}$ can send $\hat{m}$ at step $(s+1)$ in η_S^+. Consider the set of messages Rec_p received by p at s. By the induction hypothesis and the proof for Condition 4, $\hat{p}$ received a set of messages $\text{MAPM}(Rec_p)$ at step $(s+1)$.

Note that Gorilla (lines 9-11, 29) and Sandglass (lines 8-10, 24) construct message coffers in the same way based on the messages received. Let the message coffer maintained by p be M_p. Then, the message coffer $\hat{p}$ has maintained up to $(s+1)$ is $\text{MAPM}(M_p)$.

It follows that $\hat{p}$'s set $C_{\hat{p}}$ at line 11 is $\text{MAPM}(C_p)$. If the proposal value v_p of p is chosen based on the *vdf* at line 20, then $\hat{p}$ also chooses proposal value $v_{\hat{p}}$ based on a random selection at line 15. Then in η_S^+, $\hat{p}$ chooses the value v_p. Note that any $m_M \in M_p$ and $\text{MAPM}(m_M)$ have the same round number, proposal value, *priority* and *uCounter*. The variables *priority* and *uCounter* are updated the same way in Gorilla (lines 22-25) and in Sandglass (lines 17-20). Therefore, $\hat{p}$ sends message $\text{MAPM}(m)$ at s, which has $\hat{p}$ as the process, the same round number, proposal value,

priority and *uCounter* as m, and MAPM(M_p) as the message coffer.

p is a Byzantine node We further separate this case into two sub-cases:

p does not receive a peek result First, we prove that if p receives m at step $(s+1)$ in η_G^+, then $\hat{p}$ receives $\hat{m}$ at step $(s+1)$ in η_S^+. Since m is not a Peek() result, m must be sent at step s or earlier in η_G^+. By the induction hypothesis, $\hat{m}$ must be sent at step s or earlier in η_S^+. Note that in SM+ connections to defective nodes are asynchronous. Therefore, we can make the scheduler deliver $\hat{m}$ to $\hat{p}$ at $(s+1)$ in η_S^+.

Now we prove that if p generates a message m in η_G^+ at step $(s+1)$, then $\hat{p}$ can send $\hat{m}$ at step $(s+1)$ in η_S^+.

Consider $m = (r_m, v_m, priority_m, uCounter_m, M_m, nonce_m, vdf_m)$. Note that by Lemma 2, p joins the system for only one step. Therefore, for any $m_M \in M_m$, p receives it at step $(s+1)$. Therefore, from what we proved above, $\hat{p}$ receives m_M at step $(s+1)$. Therefore $\hat{p}$ has MAPM(M_m) as its coffer.

Now we prove that $\hat{p}$ can have v_m as proposal value, $priority_m$ as priority, and $uCounter_m$ as *uCounter* following the Sandglass protocol. Note that by Lemma 2, m must be valid (isValid(m) equals true).

***uCounter*$_m$ is 0** For all round $(r_m - 1)$ messages in M_m, the largest priority value for proposal values a and b are the same. Therefore, in MAPM(M_m), the largest priority value for proposal values a and b are the same. Then, it is possible that $\hat{p}$ chooses v_m as its proposal value at line 15.

***uCounter*$_m$ is not 0** All round $(r_m - 1)$ messages in M_m have

the same proposal value v_m. Therefore, in MAPM(M_m), all round ($r_m - 1$) messages also propose v_m. By Sandglass protocol (line 12), $\hat{p}$ will set its proposal value to v_m.

Since m is consistent, $\hat{p}$ will set *uCounter* to $uCounter_m$ and *priority* to $priority_m$.

Note that $\hat{p}$ joins the system for only one step, and therefore it sends only one message. We have $\hat{p}$ will send $\hat{m} = (\hat{p}, uid = 1, r_m, v_m, priority_m, uCounter_m, \text{MAPM}(M_m))$.

p receives some peek results Consider any peek result m' that p receives at step $(s + 1)$. Due to the constraint on peeking, m' does not contain any peek result, and m' is sent by a Byzantine node within step $(s + 1)$. By what we proved in the first sub-case, $\hat{m'}$ must be sent in step $(s + 1)$. Since in SM+, the defective node $\hat{p}$ can receive m'.

Then, based on the same argument as the first sub-case, we can show that $\hat{p}$ can send $\hat{m}$.

□

B.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 $\eta_S^+ = \text{INTERPRET}(\eta_G^+)$ 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 derive a contradiction.

Lemma 4. *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^+.*

Proof. Consider any execution η_G in GM and $\eta_G^+ = \text{REORG}(\eta_G)$ in GM+.

Consider a correct node p that decides a value v at step s in η_G. Consider the message $m = (r, v, priority, uCounter, M, nonce, vdf)$ that p sends right after it decides. Note that by Gorilla protocol, this is the first step that p ever collects at least $\mathcal{T}$ messages for round $(r-1)$ and $priority \geq 6\mathcal{T}+4$. By Lemma 2, p^+ receives and sends the same messages as p in the same steps. Therefore, s is also the first step that p^+ collects at least $\mathcal{T}$ messages for round $(r-1)$, and p^+ sends m at step s. Therefore, p^+ must also have decided v at step s. □

Lemma 5. *Consider any execution η_G in GM. If an execution η_S^+ in SM+ is an interpretation of an execution $\eta_G^+ = \text{REORG}(\eta_G)$ in GM+, then the following statements hold:*

1. *If a correct node p decides a value v at step s in η_G^+, then the corresponding $\hat{p}$, 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^+.*

Proof. Consider any execution η_G in GM. Consider execution $\eta_G^+ = \text{REORG}(\eta_G)$ in GM+, and η_S^+ in SM+ is an interpretation of η_G^+.

Proof for Statement (1): Consider a correct node p that decides a value v at step s in η_G^+. Consider the message $m = (r, v, priority, uCounter, M, nonce, vdf)$ that p sends right after it decides. Note that by Gorilla protocol, this is the first step that p ever collects at least $\mathcal{T}$ messages for round $(r-1)$ and $priority \geq 6\mathcal{T} + 4$. Since η_S^+ is an interpretation of η_G^+, by Definition 9, $\hat{p}$ receives the equivalent messages received by p in the same steps. Therefore, s is also the first step that $\hat{p}$ collects at least $\mathcal{T}$ messages for round $(r-1)$. Note that by definition of equivalence, $\hat{p}$ sends $\hat{m} = (\hat{p}, \cdot, r, v, priority, uCounter, M)$ at s with $priority \geq 6\mathcal{T} + 4$. Then $\hat{p}$ must also have decided v at step s.

Proof for Statement (2): Consider the first message $m = (r, v, priority = 0, uCounter = 0, M, nonce, vdf)$ that p generates for round r, and assume m is generated at step s. By the definition of equivalence, $\hat{p}$ sends $\hat{m} = (\hat{p}, \cdot, r, v, priority = 0, uCounter = 0, M)$ at s in η_S^+, and $\hat{m}$ is also the first message $\hat{p}$ sent for round r. Since $uCounter$ is 0, v is chosen based on a coin toss by Sandglass protocol. Therefore, $\hat{p}$ must have flipped a coin at step s and the result is v.

□

Theorem 2. *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 4, 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 5, $\hat{p}_1^+$ 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 1). Therefore, Gorilla satisfies agreement in GM. □

Theorem 3. *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 7, $\eta_G^+ = \eta_G$ in GM+ is trivially a reorg of η_G. Consider $\eta_S^+ = \textsc{Interpret}(\eta_G^+)$.

By the construction of the INTERPRET mapping (in Lemma 3), 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 9. Therefore, by Validity of Sandglass in SM+ (Theorem 1), no good node decides $v' \neq v$. However, by Lemma 4 and Lemma 5, p decides $v' \neq v$, which leads to a contradiction. Therefore, Gorilla satisfies validity in GM. □

B.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 [2, 4, 27]. 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 10. *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 11. *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 12. *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 13. *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.4, establishes that Byzantine nodes do not benefit from employing such a randomized strategy.

Lemma 6. *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.*

Proof. Let us fix the environment $\mathcal{E}$. Consider a randomized Byzantine strategy $\Theta_{\mathcal{E}}$ that achieves a positive non-termination probability. We omit $\mathcal{E}$ from the subscript for brevity. Denote the non-termination event in H_Θ with NT, *i.e.*, $P_{H_\Theta}(NT) > 0$. For brevity, we drop H_Θ from the subscripts for the probabilities.

We provide an inductive proof. Consider the first time that a Byzantine node takes a randomized action. This node can only choose from a countable set of actions: sending a message to a node or calculating a unit of VDF. Let us call this set of actions $\mathcal{A} = \{A_1, A_2, \dots\}$. We have $P(NT) = \sum_i P(A_i)P(NT|A_i)$. Now, since $P(NT) > 0$, there should exist an A_i such that $P(NT|A_i) > 0$, *i.e.*, the Byzantine nodes could have achieved positive non-termination probability by *determinis-*

tically taking the action A_i. Similarly, for every further randomized action with execution prefix ϕ, and with the action choices $\mathcal{A}' = \{A'_1, A'_2, \dots\}$, if $P(NT|\phi) > 0$, then there should exist an A'_i such that $P(NT|\phi, A'_i) > 0$. Repeating this process, we can carve a deterministic strategy from Θ, such that non-termination still has a positive 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

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

probability 1.

Definition 14. *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 1).

Lemma 7. *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 2, 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 REORG(H_Θ), and consider any execution η in H_Θ. By Lemma 2, correct nodes in η receive the same messages, at the same steps, as the correct nodes in REORG(η) and, moreover, the coin results in η are exactly the same as the ones in REORG(η). 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 $\text{REORG}(\eta)$. In addition, because $\text{REORG}(\eta)$ 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 4, whenever a correct node decides at some step s in η, its corresponding correct node in $\text{REORG}(\eta)$ 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. □

We can now continue our proof by showing that the INTERPRET mapping preserves the non-zero non-termination probability, which will help us prove our desired liveness property, termination with probability 1. In order to do this, we first introduce a machinery that allows us to prove the perseverance the non-zero non-termination probability throughout the INTERPRET mapping. However, first note that the INTERPRET mapping changes nothing in the executions, and only maps the "syntax" of GM+ to that of SM+. Therefore, the mapping does not introduce non-deterministic decision points for Byzantine nodes. Furthermore, as shown in Lemma 3, actions by Byzantine nodes in any execution in GM+ are translated to actions by the network in SM+. Therefore, we conclude that a Byzantine strategy Θ in GM+ is mapped to a network strategy

in SM+. Abusing notation, let us show the mapped strategy with INTERPRET(Θ).

Definition 16. *Given strategy Θ for Byzantine nodes (the scheduler message delivery), let NT_Θ be the set of executions that do not terminate. Moreover, let us define NT^i_Θ to be the event where the correct (good) node i joins, never leaves and never decides. Similarly, for every $n \in \mathbb{N}$ we define $NT^{n,i}_\Theta$ to be the event where the correct (good) node i joins, does not leave, and does not decide within the first n steps of the execution.*

First, note that we can enumerate the correct/good nodes since they are countable. Second, note that our definition of termination in Section 4.1 implies $NT_\Theta = \cup_{i=1}^{\infty} NT^i_\Theta$.

Definition 17. *Given a strategy Θ for Byzantine nodes (the scheduler message delivery) in GM+ (SM+), and a correct (good) node i, we define the random variable X^i_Θ for each $\eta \in H_\Theta$ as follows:*

$$X^i_\Theta(\eta) = \begin{cases} 1 & \text{If } i \text{ joins, never leaves, and never decides during } \eta, \\ 0 & \text{Otherwise.} \end{cases}$$

Furthermore, let us define the random variables $\{X^{n,i}_\Theta\}_{n=1}^{\infty}$ as follows:

$$X^{n,i}_\Theta(\eta) = \begin{cases} 1 & \text{If } i \text{ joins, does not leave, and does not decide within the first } n \text{ steps during } \eta, \\ 0 & \text{Otherwise.} \end{cases}$$

Lemma 23. *For every strategy Θ, every correct/good node $i \in \mathbb{N}$, and every $\eta \in H_\Theta$, we have $\lim_{n\to\infty} X^{n,i}_\Theta(\eta) = X^i_\Theta(\eta)$,* i.e., *$X^{n,i}_\Theta$ converges (almost) surely to X^i_Θ.*

Proof. Consider any execution $\eta \in H_\Theta$. If node i joins, never leaves, and never decides during η, then for every n we have $X^i_\Theta(\eta) = X^{n,i}_\Theta(\eta) = 1$.

If node i joins and leaves without deciding, then there exists a step $n_i \in \mathbb{N}$ in which the node leaves. Therefore, for every $n \geq n_i$, we have $X^i_\Theta(\eta) = X^{n,i}_\Theta(\eta) = 0$. If node i joins and decides during η, then there exists an $n_i \in \mathbb{N}$ such that node i decides at step n_i. Therefore, for every $n \geq n_i$ we have $X^i_\Theta(\eta) = X^{n,i}_\Theta(\eta) = 0$.

□

Based on Lemma 23, and the dominated convergence theorem [25], we have the following lemma.

Lemma 24. *For every strategy Θ and every node i we have* $\lim_{n\to\infty} E\{X^{n,i}_\Theta\} = E\{X^i_\Theta\}$.

Proof. Based on Lemma 23, the sequence of random variables $\{X^{n,i}_\Theta\}$ converges pointwise to X^i_Θ. Furthermore, it is clear that every $X^{n,i}_\Theta$ is non-negative and bounded from above by 1. This satisfies the conditions required for the dominated convergence theorem, thus the theorem tells us $\lim_{n\to\infty} E\{X^{n,i}_\Theta\} = E\{X^i_\Theta\}$. □

Lemma 25. *For every strategy Ψ in GM+, there exists a strategy Λ in SM+ such that* $H_\Lambda = \{\text{INTERPRET}(\eta) | \eta \in H_\Psi\}$.

Proof. We show that the strategy Λ exists, and is in fact the same as Ψ. For brevity, let I_Ψ denote $\text{INTERPRET}(H_\Psi)$, and consider any execution η in H_Ψ. According to Lemma 3, correct nodes in η receive the same messages, at the same steps, as the good nodes in $\text{INTERPRET}(\eta)$. Thus, the message history up to any step s in η is the same as the message history up to the same step in $\text{INTERPRET}(\eta)$. In addition, since $\text{INTERPRET}(\eta)$ is an actual SM+ execution, compatibility is trivially satisfied. Thus, we conclude that the executions in I_Ψ follow the same strategy as in Ψ. Naming this strategy Λ finishes the proof.

□

Lemma 8. *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 strategy $\Lambda_{\mathcal{E}'}$ in SM+ that also achieves a positive non-termination probability.*

Proof. Let us assume that there exists an environment $\mathcal{E}$ and a Byzantine strategy $\Theta_{\mathcal{E}}$ in the Gorilla protocol that leads to a positive non-termination probability. According to Lemma 7, there exists a strategy Ψ in GM+ that achieves a positive non-termination probability, in environment $\mathcal{E}$. Consider the INTERPRET mapping. Since, according to Lemma 3, the joining/leaving and initial value schedules for correct nodes are bijectively mapped by the INTERPRET mapping to the joining/leaving and initial value schedules of the good nodes, respectively, we just set $\mathcal{E}' = \mathcal{E}$. In the rest of the proof, we omit the environments for brevity.

First, note that according to Lemma 5, the INTERPRET mapping preserves all of the coin tosses in H_Ψ. Moreover, for a given execution $\eta \in H_\Psi$, the same lemma tells us that INTERPRET(η) might include *more* coin tosses than those in η.

Consider Λ = INTERPRET(Ψ), based on Lemma 25. Let us also define the events NT_Ψ, NT_Ψ^i, $NT_\Psi^{n,i}$, NT_Λ, NT_Λ^i, and $NT_\Lambda^{n,i}$ based on Definition 16, for the strategies Ψ and Λ. Since $P_{H_\Psi}(NT_\Psi) = P_{H_\Psi}(\cup_{i=1}^{\infty} NT_\Psi^i)$, using the union bound we have $P_{H_\Psi}(NT_\Psi) \leq \sum_{i=1}^{\infty} P_{H_\Psi}(NT_\Psi^i)$. Now, Ψ achieves a positive non-termination probability, *i.e.*, $P_{H_\Psi}(NT_\Psi) > 0$. Therefore, there should exist some $i^* \in \mathbb{N}$ such that $P_{H_\Psi}(NT_\Psi^{i^*}) > 0$, since otherwise we would have $P_{H_\Psi}(NT_\Psi) = 0$ based on the union bound. We now define the random variables $X_\Psi^{i^*}$ and X_Ψ^{n,i^*} as in Definition 17, and based on Lemma 24 we have $\lim_{n\to\infty} E\{X_\Psi^{n,i^*}\} = E\{X_\Psi^{i^*}\}$. Similarly, we define the random variables $X_\Lambda^{i^*}$ and X_Λ^{n,i^*}, and once again Lemma 24 tells us

$\lim_{n\to\infty} E\{X_{\Lambda}^{n,i^*}\} = E\{X_{\Lambda}^{i^*}\}$.

Based on Definition 16, we know that for every n, we have $NT_{\Psi}^{n,i^*} \subset NT_{\Psi}^{n+1,i^*}$ and $NT_{\Lambda}^{n,i^*} \subset NT_{\Lambda}^{n+1,i^*}$. Therefore, we have $E\{X_{\Psi}^{n,i^*}\} = P_{H_\Psi}(NT_{\Psi}^{n,i^*}) \leq P_{H_\Psi}(NT_{\Psi}^{n+1,i^*}) = E\{X_{\Psi}^{n+1,i^*}\}$ and $E\{X_{\Lambda}^{n,i}\} = P_{H_\Lambda}(NT_{\Lambda}^{n,i^*}) \leq P_{H_\Lambda}(NT_{\Lambda}^{n+1,i^*}) = E\{X_{\Lambda}^{n+1,i^*}\}$. Since the increasing sequence $\{E\{X_{\Psi}^{n,i^*}\}\}_{n=1}^{\infty}$ converges to $E\{X_{\Psi}^{i^*}\} = P_{H_\Psi}(NT_{\Psi}^{i^*}) > 0$, there should exist a step n^* such that $P_{H_\Psi}(NT_{\Psi}^{n^*,i^*}) = E\{X_{\Psi}^{n^*,i^*}\} > 0$. Now, given n^*, let us consider $P_{H_\Lambda}(NT_{\Lambda}^{n^*,i^*}) = E\{X_{\Lambda}^{n^*,i^*}\}$. This value is computed based on the first n^* steps of the executions in H_Λ. Based on Lemma 5, we know that these executions contain all of the coin tosses happening in the first n^* steps of the corresponding executions in H_Ψ. Moreover, they might contain more coin tosses as explained above. Therefore, if the probability of the event $NT_{\Psi}^{n^*,i^*}$ in H_Ψ is positive, the probability of the corresponding event $NT_{\Lambda}^{n^*,i^*}$ in H_Λ should also be positive, *i.e.*, $E\{X_{\Lambda}^{n^*,i^*}\} > 0$. Since the sequence $\{E\{X_{\Lambda}^{n,i^*}\}\}_{n=1}^{\infty}$ is increasing with n and converging to $E\{X_{\Lambda}^{i^*}\}$, we should therefore have $E\{X_{\Lambda}^{i^*}\} > 0$. It immediately follows that $P_{H_\Lambda}(NT_{\Lambda}^{i^*}) = E\{X_{\Lambda}^{i^*}\} > 0$.

Finally, since $NT_{\Lambda}^{i^*} \subset NT_{\Lambda}$, we should have $0 < P_{H_\Lambda}(NT_{\Lambda}^{i^*}) \leq P_{H_\Lambda}(NT_{\Lambda})$. This means that Λ is a scheduler strategy that achieves positive non-termination probability, and finishes our proof.

□

Theorem 4. *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 7, there exist a GM+ environment and a strategy Ψ for the Byzantine nodes in GM+ that achieve a positive non-termination probability. Simi-

larly, by Lemma 8, 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 1). Thus, Byzantine strategy Θ cannot force a positive non-termination probability; Gorilla terminates with probability 1. □

BIBLIOGRAPHY

[1] Ittai Abraham, Dahlia Malkhi, et al. The blockchain consensus layer and bft. *Bulletin of EATCS*, 3(123), 2017.

[2] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2):165–175, 2003.

[3] 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.

[4] Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. *Foundations of probabilistic programming*. Cambridge University Press, 2020.

[5] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pages 27–30. ACM, 1983.

[6] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[7] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[8] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[9] Chao Chen and Fangguo Zhang. Verifiable delay functions and delay encryptions from hyperelliptic curves. *Cybersecurity*, 6(1):54, 2023.

[10] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[11] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the*

Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I 25, pages 248–277. Springer, 2019.

[12] 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.

[13] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and Nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020.

[14] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[15] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[16] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[17] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1982.

[18] Forbes. Crypto prices, 2024. Accessed: 2024-06-14.

[19] Eli Gafni and Giuliano Losa. Brief announcement: Byzantine consensus under dynamic participation with a well-behaved majority. In *37th International Symposium on Distributed Computing (DISC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[20] Eli Gafni, Michael Merritt, and Gadi Taubenfeld. The concurrency hierarchy, and algorithms for unbounded concurrency. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 161–169, 2001.

[21] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone

protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[22] 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.

[23] Dongning Guo and Ling Ren. Bitcoin's latency–security analysis made simple. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 244–253, 2022.

[24] Maurice Herlihy. Blockchains and the future of distributed computing. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC '17)*, page 155, August 2017. Keynote Address.

[25] Jean Jacod and Philip Protter. *Probability essentials*. Springer Science & Business Media, 2004.

[26] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.

[27] Benjamin Lucien Kaminski. *Advanced weakest precondition calculi for probabilistic programs*. PhD thesis, RWTH Aachen University, 2019.

[28] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

[29] Rami Khalil and Naranker Dulay. Short paper: Posh proof of staked hardware consensus. *Cryptology ePrint Archive*, 2020.

[30] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

[31] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 729–744, 2018.

[32] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative Byzantine fault tolerance. *Communications of the ACM*, 51(11):86–95, November 2008.

[33] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[34] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[35] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[36] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[37] Andrew Lewis-Pye and Tim Roughgarden. Byzantine generals in the permissionless setting. *arXiv preprint arXiv:2101.07095*, 2021.

[38] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. Xft: Practical fault tolerance beyond crashes. In *OSDI*, pages 485–500, 2016.

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

[40] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[41] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy bft. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 490–503, 2023.

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

[43] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. *Cryptology ePrint Archive*, 2022.

[44] Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. In *Annual International Cryptology Conference*, pages 381–409. Springer, 2019.

[45] Achour Mostefaoui, Michel Raynal, and Frédéric Tronel. From binary consensus to multivalued consensus in asynchronous message-passing systems. *Information Processing Letters*, 73(5-6):207–212, 2000.

[46] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01.

[47] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

[48] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

[49] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019.

[50] Moritz Platt and Peter McBurney. Sybil in the haystack: A comprehensive review of blockchain consensus mechanisms in search of strong sybil attack resistance. *Algorithms*, 16(1):34, 2023.

[51] Michael O Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983.

[52] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[53] Yee Jiun Song and Robbert van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *International Symposium on Distributed Computing*, pages 438–450. Springer, 2008.

[54] TK Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[55] Benjamin Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on*

the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38, pages 379–407. Springer, 2019.

[56] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[57] 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.

[58] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.