The Spacemesh Protocol: Tortoise and Hare Consensus... In... Space

Iddo Bentov Julian Loss Tal Moran Barak Shani

September 13, 2019

1 Introduction

In the short period since their inception a bit more than a decade ago, cryptocurrencies have already had a significant impact. Inspired by this success, a flurry of alternative cryptocurrencies has emerged, many of them with the goal of addressing major unsolved problems that exist in the original Bitcoin[25] design and other deployed systems.

The blockchain structure that Bitcoin introduced is simple and elegant, and most other cryptocurrencies also use this same structure. However, a blockchain has some inherent deficiencies. The consensus guarantee in Bitcoin-like blockchain protocols relies on having frequent time intervals in which only a single block is created, and the length of this interval is lower bounded by the network latency. This inherently limits the transaction volume that such a blockchain can support, as well as the frequency at which miners can be rewarded for generating blocks.

In practice, this causes higher transaction fees, and incentivizes miners to form centralized pools in order to reduce the expected time and variance until they earn a reward. Moreover, Bitcoin-like blockchain protocols are not incentive-compatible under certain conditions, i.e., extending the longest chain is not necessarily the rational behavior [28, 5, 13, 31].

To overcome these problems, cryptocurrency protocols that rely on a directed acyclic graph (DAG) structure instead of the chain topology have been proposed. For instance, Phantom [34] and Meshcash [5] are protocols in which miners perform PoW computations to create blocks of a DAG.

Ideally, the reliance on PoW should also be eliminated. One reason is that PoW in itself creates centralization pressures due to an advantage of specialized hardware (ASIC) over general-purpose computation devices [6], implying higher profitability for miners with easy access to manufacturing plants and miners located in areas with cheap electricity. Perhaps of greater concern, PoW-based cryptocurrencies have unavoidable environmental costs: the security of these systems requires miners to continuously generate proofs of work at their maximum computational capacity, which requires increasing energy expenditure as the number of miners in the system grows. In fact, mining-related power consumption worldwide currently matches the electricity usage of a medium sized country [30].

1.1 Our Results

In this work we propose Spacemesh, a novel consensus protocol that addresses these concerns. The Spacemesh protocol is fully permissionless (with a low barrier to entry for new miners), supports high transaction throughput and requires far less energy than PoW-based protocols. The highlights of our construction are:

- Spacemesh replaces CPU work as the underlying limited resource with *spacetime*—disk storage over time. More specifically, we replace PoWs with Proofs of SpaceTime (PoSTs) [24]. Since disk storage does not require energy use when at rest, the total energy cost is much lower.
- Spacemesh is based on a *mesh* (layered DAG) topology rather than a chain. That is, instead of blocks being generated effectively one at a time, Spacemesh miners generate multiple blocks in

parallel—and they can all be "accepted" as valid blocks. The base protocol builds on the Meshcash framework [5], but makes non-trivial modifications to support spacetime as the underlying resource.

- The state of the system is a deterministic function of the *contents* of the mesh. In particular, the state doesn't depend on the *timing* of received messages. This property ensures that (1) new users joining the network can agree on the "correct" state as long as they can communicate with a *single* honest miner and (2) consensus is immediate if the system is not under attack.

- The Spacemesh protocol can "self-heal" from an arbitrary violation of our security assumptions—even if the system is under continuous attack from an adversary controlling a constant fraction of the spacetime resources. That is, honest parties will converge to consensus from any initial condition, once the security assumptions are satisfied again. (Self-healing does require a period of "relative" quiet—i.e., a period during which the adversary attacks with $q \ll 1/3$ of the spacetime resources.)

- Spacemesh is designed to be a *race-free* protocol—honestly generated blocks always become valid, implying that a powerful (perhaps adversarial) miner cannot receive a disproportionately high portion of the rewards. This makes it much easier to prove the protocol is incentive-compatible.

- Spacemesh is designed to have a high degree of decentralization: the high throughput of blocks implies that each individual miner can be rewarded frequently (no need for pooled mining), the race-freeness implies the honest behavior is always rewarded, and the spacetime resource is readily available to home users—this increases the likelihood that many individual miners will join the system.

1.1.1 Security Guarantees

We prove that the Spacemesh protocol is secure as long as the adversary controls less than a $q < 1/3$ fraction of spacetime resources in the system, and under reasonable network synchrony assumptions (specifically, we assume that every message seen by an honest party at time t will be seen by *all* honest parties at time $t + \delta$, where time is measured in "rounds"; for concreteness, one can think of δ as 30 seconds, but the exact number depends on empirically measured network latency).

If the conditions are temporarily violated (i.e., the network is asynchronous or the honest miners are a minority), then the Spacemesh protocol will still self-heal when these assumptions hold again: after the self-healing period (whose length depends on the duration and severity of the violations), the honest miners are guaranteed to be in consensus on the entire history.

Note that Bitcoin also requires a synchronous network (there cannot be agreement on the longest chain during a network partition, see also [27]), and self-heals in a similar manner, but prior security proofs [27, 15] do not analyze the self-healing property (including analysis that assumes variable difficulty [16]). By contrast, we give a rigorous security proof of the self-healing guarantee of Spacemesh, in a setting where the total amount of spacetime resources is variable.

1.2 High-Level Overview

In this writeup we concentrate on the construction of a permissionless, decentralized ledger. This can be used as the underlying consensus mechanism for a cryptocurrency. For a complete cryptocurrency, one would also need to describe the incentive mechanism design and the specifications of the "consensus computer"—the protocol that determines how transactions are parsed and how the cryptocurrency's state evolves.

Since our consensus protocol is secure against malicous adversaries and not just rational ones, we can give a self-contained description of the ledger while deferring the details of the higher layers of the cryptocurrency.

1.2.1 Basic Protocol Terms

The Spacemesh protocol's goal is to achieve consensus on an ordered set of transactions, and ensure this set is "append-only". For the purposes of this protocol, we treat transactions as opaque strings (the semantic meaning of a transaction is determined by the consensus-computer layer). We explicitly allow transactions to appear multiple times in the ledger—this is without loss of generality, e.g., the consensus computer can choose to consider only the first appearance of each transaction in the ledger. (An implementation might choose to restrict which transactions can enter the ledger in the first place in order to increase efficiency, but for simplicity we ignore such optimizations in this writeup.)

Blocks and Layers Like most existing blockchain protocols, we group transactions into *blocks*; each block includes an ordered set of transactions. Unlike most block*chain* protocols, the Spacemesh protocol expects multiple blocks to be published in parallel. Moreover, block publication times are deterministic rather than stochastic: a *layer* of blocks is published at fixed intervals (e.g., one layer every five minutes). The protocol specifies that all blocks in the same layer are published at the same time. Every block explicitly contains its *layer index*—the layer at which the block claims to have been published—and a unique block ID (think of this as a collision-resistant hash of the block contents). (Section 4.5 contains a detailed specification of a block.)

Constructing a *Block* Ledger is Sufficient Our protocol actually constructs a "block ledger"—an ordered, append-only set of blocks—rather than a transaction ledger. However, a block ledger can easily be viewed as a transaction ledger by iterating over the blocks in their ledger order, and for each block iterating over its included transactions. For transactions that are duplicated identically in multiple blocks (this situation is unavoidable if we allow blocks to be published in parallel), we consider only the earliest appearance of the transaction as part of the transaction ledger.

Deciding Block Validity is Sufficient If we achieve consensus on the *validity* of each block, we immediately get a block ledger. This is because the blocks explicitly contain their layer index, so we can order the valid blocks by layer, and within each layer by block ID. As long as all honest parties agree on the validity of blocks, this implies that they also agree on the order of valid blocks. (To guarantee the "append-only" property of the ledger, we also need to ensure that valid blocks cannot become invalid, and vice versa.) Thus, we reduce the problem of constructing a transaction ledger to that of deciding the validity of each block separately. See section 4.1 for a detailed description of the reduction.

Syntactic and Contextual Validity We distinguish between two "types" of validity rules for a block. *Syntactic* validity rules depend only on the contents of a block (and any blocks it references). The important property of syntactic validity is that once a block is considered syntactically valid it will be syntactically valid forever after (since its contents are fixed). Moreover, every two honest parties who receive the same block will always agree on its syntactic validity.

Unfortunately, this cannot be the only type of validity rule. This is because an adversary can always create a syntactically-valid block in the far past—since it cannot be forced to forget the information it knew in the past, it can simulate itself in the past running the honest block-generation protocol. If we allow such blocks to be considered valid the adversary can use this strategy to change the history of the ledger.

Thus, we require a second type of validity rule, which we call *contextual* validity. Contextual validity rules encompass all validity rules that are not syntactic. In particular, contextual validity can depend on information received *after* a block was generated (e.g., the longest-chain rule in Bitcoin) and even on local timing information (when a block was received according to a local clock).

One of the goals of the Spacemesh consensus protocol is to depend only on message contents to determine validity (this makes consensus far more robust, since timing information is not verifiable); thus, our contextual validity rules do *not* depend on timing information. See sections 4.7 and 4.8 for the formal description of the Spacemesh validity rules.

1.2.2 Challenges in Achieving Consensus

We've established that the purpose of our consensus protocol is to ensure that honest parties agree on the (contextual) validity of blocks. Our protocol assumes the existence of a gossip network, which guarantees that every message sent or received by one honest party is eventually received by all honest parties. If all honest parties see the same messages, why don't we trivially get consensus?

The reason is that honest parties may not agree on the *timing* of messages. However, timing is critical. For example, for every layer i, there must be some point in time after which we can no longer accept a block that claims to be in layer i. Unfortunately, no matter where we set the threshold, an adversary can always publish a block close enough to the threshold time that some of the honest parties receive it before the threshold, while some receive it afterwards—causing the honest parties to disagree on the validity of the block.

Moreover, if the protocol specifies that parties should send additional messages in order to resolve the disagreement, the timings of the additional messages can *also* be a source of disagreement. If the protocol isn't designed carefully, it can allow an adversary to succeed in a "balancing attack", which requires very few resources but prevents honest parties from reaching consensus *indefinitely*.

1.2.3 Consensus by Election

Our consensus protocol consists of two subprotocols: the *Tortoise*, which guarantees "slow but sure" eventual consensus and the irreversibility of the ledger (see section 4.9 for a more detailed explanation and sections 4.10 and 4.11 for Tortoise optimizations), and the *Hare*, which allows very fast convergence (see section 6 for more details).

The underlying intuition for the Tortoise protocol is an "election". Think of miners as voters; when a miner generates a block, it includes, in addition to the transaction data, a vote for every previous block that the miner believes is contextually valid.

Miners decide how to vote about a block B in the past by counting the votes for and against it (any block generated after B that didn't vote for it is considered to vote against it). Like most democratic elections, the miners accept the majority opinion. Unlike most democratic elections, votes are *weighted*, with the weight of each vote proportional to the spacetime resource it "represents".

Irreversibility If all honest parties agree that a majority of votes supports the validity of a block B at time t, then, with high probability, all honest parties will continue to support its validity. The proof is very similar to the irreversibility argument in Bitcoin: honest miners will vote in support of B as long as the majority of votes support it, but because honest miners control more than half of the spacetime resource (by our "honest majority" assumption), the supporting votes will be generated at a greater rate than opposing votes, so w.h.p. the vote margin in favor of B only grows with time.

1.2.4 Bootstrapping Consensus With the Hare

While the simple election mechanism works for blocks that are far enough in the past, it clearly doesn't work for blocks that have just been published—there are no votes to count yet (since blocks in the same layer are published at the same time, they can't be expected to know about each other). Moreover, the irreversibility argument above requires all honest parties to agree on validity in the first place.

This is where the Hare protocol enters the picture. The Hare protocol is a fast, permissionless byzantine agreement protocol. An instance of the Hare protocol is executed after each layer is published, and its output is the set of valid blocks for that layer. This serves as a bootstrapping protocol for the Tortoise election: all honest miners use the output of the hare protocol when determining how to "vote" in the Tortoise election for recent layers, and use the "majority rule" for older layers. The properties of the byzantine agreement protocol guarantee that every block that is rejected by *all* honest parties will not appear in the output (e.g., blocks that are extremely late will always be considered invalid), while every block that is accepted by all honest parties will always appear in the output (so honest blocks will always be valid).

The Hare protocol runs over the gossip network, but only the *results* of the protocol—as reflected in the miners' votes—are recorded on the blockmesh. The protocol execution itself does not need to be stored.

1.2.5 Self-Healing With the Tortoise

The ideas described above suffice to construct a fast and secure consensus protocol. However, this protocol is "fragile", in the sense that its security depends on all of our underlying assumptions holding *continuously* since genesis. For example, if at layer i the hare protocol fails (e.g., due to some very low probability event), honest parties may not agree on the validity of the blocks in layer i. Suppose honest parties are evenly split about the validity of some block B in layer i—half accept it as valid and half don't. In subsequent layers, the vote margin for B will be very close to 0—and an adversary can succeed in a *balancing attack*: the adversary keeps the margin small *indefinitely* by always voting "with the minority"; since the margin is already small, the adversary does not have to expend many resources in order to "pad" the minority so that the total number of positive and negative votes remains equal. Thus, honest parties will never reach consensus on B's validity.

Adding a Random Ingredient Our solution to this is a simple modification of the Tortoise protocol that allows it to reach consensus from *any* initial state. The main idea is to randomize honest parties' votes when the vote margin is small—but in a coordinated way. For intuition, suppose we had a sequence of coin flips that is:

- *Public*: at the start of layer $i + 1$, all honest parties agree on the result of the i^{th} coin flip.
- *Independent and Unbiased*: each coin comes up heads and tails with equal probability.
- *Unpredictable*: the adversary cannot guess the result of the i^{th} coin flip until the end of layer i.

A Coin-Flipping Tortoise In our modified Tortoise protocol, instead of always voting using the majority vote, the honest miner considers two different cases: (1) If the vote margin is "large", the miner votes according to the majority. (2) If the vote margin is "small", the miner instead votes according to the result of the coin flip.

We set the threshold for "large" by bounding how much honest miners can disagree—in particular, we want to ensure that two honest miners cannot see a "large" voting margin in opposing directions. At the same time, if *all* honest parties agree on the validity of B and vote appropriately, in the next round all honest parties should have a vote margin that is "large".

These properties are sufficient to achieve our goal. If the margin ever becomes "large", all honest parties will agree in the next round, so our original irreversibility argument holds again. On the other hand, for every layer in which at least one honest party sees a "small" margin, the honest parties that see a "large" margin must all vote in the same direction. If the coin flip for that layer is also in the same direction (or in any direction if all honest parties have "small" margins), all honest parties will vote in the same direction). Since the coin flips are independent and unbiased, this happens with probability at least $1/2$ in each layer, so the honest parties will reach consensus very quickly.

A "Weak" Coin is Sufficient The same idea still works even if our sequence of coin flips is not perfect—for example, if the adversary *can* guess the result of the coin flip early some of the time, or bias the result so that heads and tails are no longer equally likely.

We prove that a "weak" coin is sufficient: as long as the adversary can't predict the result with more than constant probability, and can't bias it "too far", the same basic analysis will guarantee eventual convergence (the expected time it takes to reach consensus does grow as the coin "weakens", however). Moreover, we don't need the coin to be publicly verifiable: it's enough that honest parties agree on its value—they don't need to *prove* that this value is correct.

Being able to use a "weak" coin is important, since it is much easier to construct compared to a strong one. We discuss possible constructions in section 4.3.

1.2.6 Proofs of Spacetime

In a permissionless protocol, there is no trusted identity verification, so such protocols are vulnerable to *Sybil attacks*, in which the adversary creates and controls many "fake" identities. To overcome this, we use a limited, publicly-verifiable resource in place of identities. As we explained above, the resource used by the Spacemesh protocol is spacetime.

We make spacetime a *publicly-verifiable* resource by having miners publish Proofs of Spacetime (PoSTs). At a high level, a PoST is a proof that a node allocated a certain amount of *space* S toward participating in the mining process over a given period of *time* T. The node's spacetime resource is computed as $S \cdot T$.

Loosely speaking, the PoST consists of two phases: an initialization phase (executed once), in which miners "commit" to the data that fills the space S, and an execution phase (executed repeatedly), in which miners prove that they are still storing the data. The time component of the spacetime resource is the elapsed time between successive proofs—if the interval between initialization (or the previous execution phase) and the latest execution phase is T, this proves the miner expended $S \cdot T$ spacetime.

Unfortunately, a PoST can't actually prove that the miner *stored* the data between one proof and the next. It proves a slightly weaker statement: "either the miner stored the data, *or the miner reconstructed the data*". This is unavoidable, since a miner can always rerun the initialization procedure to recreate the data.

We handle this by explicitly parameterizing the initialization cost in a PoST.[1] The initialization cost is important because its relation to the cost of storage determines whether it is *rational* to store the data in the interval between proofs, or whether to recompute instead. If the initialization cost is lower than the cost of storing the data, rational users will prefer to recompute—in this case, the protocol remains secure, but essentially devolves to one based on proof-of-work. Since the real-world costs of both storage and CPU can fluctuate, we must be able to adjust the initialization cost in order to ensure that storing data remains the rational choice.

Moreover, in the Spacemesh protocol, our solution to maintaining fixed communication complexity is to increase the interval between proofs as the number of miners grows (see section 1.2.7). This means the cost of storing the data between proofs grows linearly with the number of miners,[2] which will eventually require adjustment of the initialization cost even if the costs of CPU and storage are fixed.

Proofs of Elapsed Time and Non-Interactive PoSTs While the space component of the PoST is publicly verifiable—it depends only on the contents of the messages sent in the PoST protocol—the *time* component is not: it requires the verifier to measure the elapsed time between PoST executions.

We convert the PoST into a fully "non-interactive", publicly-verifiable primitive (a NIPoST) by adding a Proof of Elapsed Time (PoET) to the construction. Intuitively, the miner will use the PoET to prove in a publicly-verifiable way that an interval of length T has elapsed between the PoST executions. To verify that the miner used $S \cdot T$ spacetime, it is sufficient to check that the PoST is for S space and the PoET for T time.

Since we don't have a way to directly prove that time has elapsed, we use *sequential work* as a proxy for elapsed time (think of sequentially iterating a cryptographic hash). The underlying idea is that it's extremely hard to make the computation of an iterated sequence of hashes faster that the fastest mass-produced commercial CPU, in particular if we use a hash (such as SHA256) that mainstream CPU makers have already spent considerable resources accelerating. (This is in sharp contrast to increasing *total* work throughput—that can be done by parallelizing, at a cost that's merely linear in the desired throughput.). Thus, in this paper we use PoET and PoSW (Proof of Sequential Work) interchangeably.

See section 4.2.1 for an in-depth description of how the NIPoST is constructed.

[1] All current Proof-of-Space protocols implictly rely on costly initialization as well, but because it is fixed, it does not show up in their definitions.

[2] Note that this change is not *continuous*—we use expontentially growing "steps" to avoid frequent difficulty adjustments.

1.2.7 Reducing Communication Overhead

The security of the Spacemesh protocol depends on honest miners controlling a majority of the spacetime resources, and thus controlling a majority of the "voting weight" in the Tortoise election, and a majority of the participants in the Hare protocol (all byzantine agreement protocols require an honest majority [**?**]).

However, we wouldn't want every miner to create a block in every layer, or participate in every Hare protocol instance—the communication and storage costs of such a solution would be completely impractical.

Existing blockchain protocols solve the communication overhead problem by *randomly sampling* a set of miners from the total population (e.g., one can think of the Bitcoin PoW as a mechanism to "sample" a random miner out of the entire miner population). This is useful for the same reason that election polls are: the sample size can be much smaller than the population size (it depends only on the security parameter]) while still ensuring that the sample is "representative" with high probability.

However, it also has drawbacks. In particular, it is vulnerable to attacks on the randomness used to select the sample. This isn't just a theoretical issue: existing methods of generating unbiased, publicly-verifiable, randomness all have significant problems. Moreover, the solution used in PoW-based protocols is counterproductive in our case: they use the proof of work itself as the source of randomness. Essentially, every party can bias the result proportionally to the amount of work they expend—which is exactly the result they wish to achieve. However, in Spacemesh we need to ensure that extra work cannot bias the result too much, otherwise it would devolve into a PoW-based protocol—this is called the "grinding" problem.

Deterministic Proofs of Resource Consumption Spacemesh sidesteps the problem by requiring *every* miner to publish a NIPoST proof at deterministic intervals (called *epochs*). Since there is no sampling, grinding cannot help increase a miner's weight. In order to keep the communication overhead constant, we increase the epoch length as the number of miners increases, so that the number of proofs per time interval remains unchanged.[3]

In Spacemesh, the NIPoST proof is part of a special transaction called an *Activation Transaction* (ATX). Each miner publishes one ATX per epoch (for simplicity, if a miner controls multiple space units we consider them multiple miners, each controlling a single unit) and *every* miner that publishes an ATX will be allowed to generate blocks in the succeeding epoch (the total voting weight of these blocks will be the spacetime of the NIPoST).

"Unverifiable" Sampling While miners are guaranteed a block in each epoch, we do randomize the position of the block within the epoch. The randomization ensures that (with high probability) the majority of blocks in every layer is generated by honest parties (so that the Tortoise "election" will always give the correct results).

In order to do this, we still need a source of randomness—a *random beacon*—that is unpredictable and cannot be biased too much by the adversary. However, we *don't* require the beacon to be publicly-verifiable: it's enough that honest parties that are online at the time agree on its value. In the Spacemesh protocol, miners "self-report" their randomness. Honest miners will (by our beacon construction) report the same value, but the adversary can report an arbitrary value. Since the beacon is used for determining when in the epoch the miner is eligible to publish a block, the adversary can use this beacon to concentrate blocks in a small interval.

We use a novel idea to deal with this kind of attack: honest parties "ignore" votes cast by blocks that don't match their own beacon value in the current epoch. This ensures means that the adversary can only lose voting power by picking a different beacon value. For previous epochs, the concentration of blocks doesn't matter—the Tortoise protocol will count all the blocks in the epoch, and for an entire epoch the

[3] In practice, the initial two-week epoch would suffice for over 800000 miners, so epoch length will probably grow very slowly.

honest miners are guaranteed to have the correct fraction of the total weight.[4] We analyze this formally in lemma 5.4.

We also use a random beacon to select miners for other roles, such as participation in "Hare committees". Although the requirements for the hare committee selection are slightly different (and hence we rely on random beacons with different parameters), we use the same general technique to deal with an adversary using a "fake" beacon value: the honest parties simply ignore messages using beacon values that are different from their own. Since all honest parties agree on their beacon value, for the adversary, choosing a different value is equivalent to simply aborting.

2 Related Work

2.1 Spacemesh compared to Meshcash

Spacemesh is based on the Meshcash "Tortoise and Hare" framework. However, there are several major design choices that make Spacemesh fundamentally different than Meshcash:

- PoWs "bind" the expended CPU work to a specific challenge. Existing PoW-based protocols (including Meshcash) make strong use of this property; it ensures that the adversary cannot reuse work it has already done to create an "alternate history". PoSTs, in contrast, do not bind the expended space-time to a challenge (since we want to be able to reuse the stored data for multiple challenges in order to keep the energy cost low). This means an adversary can create "syntactically-valid" blocks that reuse "old" space-time—which is something the protocol must be able to deal with.

- The times at which solutions to PoWs are found have a Poisson distribution. This feature is critical in securely implementing the random sampling of miners in Meshcash (as well as other PoW-based protocols). In contrast, Spacemesh replaces the lottery with *deterministic* eligibility criteria: *every* miner that expends a sufficient space-time resource will be eligible to generate a block (there is some randomization for exactly *when* the block is generated). Since eligibility isn't randomized, Spacemesh is much less vulnerable to "grinding" attacks (where the adversary attempts to perform extra work—that is not specified by the honest protocol—in order to increase their probability of being selected in the lottery).

2.2 Spacemesh compared to Proof of Stake protocols

Proof of Stake (PoStake) based cryptocurrencies require a scarce resource that is not physical in nature – i.e., the resource is digital money that circulate within the system. As such, PoStake is even greener than Spacemesh, but there are important deficiencies of PoStake that Spacemesh eliminates.

For one, PoStake protocols compel miners to lock security deposits in order to mitigate nothing-at-stake equivocation attacks. If a substantial pre-allocated deposit is required then the protocol is akin to a centralized (committee based) consensus protocol, since each participant in the consensus protocol pays the opportunity cost by locking a pre-allocated deposit for an extended period of time. This implies a high barrier to entry into the mining market (dPoS [] is an acute example). If there is no requirement for a pre-allocated deposit then the confirmation time has to be larger, since dishonest miners may pretend to be inactive and later equivocate – they cannot be penalized for such attacks because there are no locked funds to confiscate. By contrast, the Spacemesh consensus protocol has no security deposits, and its confirmation time is still much faster than that of Bitcoin.

PoStake protocols are also not secure against long-range attacks, because of an inherent costless simulation [] problem. Spacemesh is substantively different in this regard: NIPoSTs in old layers can be equivocated (making them contextually invalid), but this does not give old miners the power to create

[4] This isn't quite accurate—the adversary *can* get some advantage, but the basic intuition holds.

any blocks in newer layers. Thus, the costless simulation concern is far less significant in Spacemesh than in PoStake protocols.

Formally, PoStake cannot be regarded as permissionless consensus mechanism, since newcomers cannot join unless a current coin holder agrees to perform a transaction that transfers some of her coins. Spacemesh, on the other hand, is truly permissionless: a newcomer that expends a space-time resource to broadcast an activation transaction does not need to be included in blocks of the current timeframe. While formal permissionlessness may be of little interest, the potential for cartel censorship is a real concern in PoStake protocols. That is, a majority among the stakeholders who are currently eligible to participate in the consensus protocol may take over the system and prevent everyone else from contributing to the consensus, and may then engage in subtle attacks such as prioritizing transactions according to out-of-bound payments. This concern is even greater at an early stage of the network as the takeover cost is low—the corrupt cartel may ensure that it remains a majority while allowing others to participate in the consensus, and begin to mount active attacks only after the network grew to become more valuable.
.

There are also practical implications that relate to permissionlessness:

- Purchasing new and possibly obscure cryptocurrency can be burdensome for people in developing countries, and even in developed countries.
- Storage equipment is a general-purpose consumer-level item that can be readily obtained. Furthermore, it is highly common that storage devices are underutilized, e.g., it is not unusual for an hard drive in a new computer to be half empty. Thus, home users already have the resource needed to become Spacemesh miners and earn rewards, as opposed to PoStake (and PoW) miners that must obtain the mining resource in advance. Since many home users may participate with zero marginal cost, there is a particularly good potential for decentralization in a space-time based system.

Beside the differences at the level of principle, there are real-world problems with PoStake systems that do not apply to PoST. Specifically, many users transfer their coins to wallets of exchange services, assuming a certain risk in order to perform real-time trades against fiat currencies and other cryptocurrencies. As a result, the exchange operator becomes an extremely large coin holder in the system, and would have much power in a PoStake consensus protocol. Thus, while the intention of the exchange users is simply to deposit their funds for the purpose of real-time trading, they unwitting give away a lot of decision-making power w.r.t. the consensus system itself.

2.3 Leader-Election-Based PoW Based Protocols

GHOST The GHOST protocol of Zohar and Sompolinsky [33] modifies the Bitcoin best-chain rule to take into account valid blocks that are not on the longest chain. This means that less honest mining power goes to waste, and therefore GHOST can support a shorter interval between blocks without sacrificing security (cf. [20] for a formal analysis). The main motivation behind GHOST is to increase the transaction throughput. While this is also one of the goals of Spacemesh, several of our other concerns – such as race-freeness and incentive-compatibility – are not addressed in the GHOST protocol (e.g., GHOST is vulnerable to selfish mining attacks). With regard to frequent payouts to small miners, GHOST improves upon Bitcoin since it allows an increase in the block generation rate, but the growth rate of the main chain (i.e., the payout rate) is still bounded as a function of the network propagation time. In fact, the main chain in GHOST grows at a slower pace than what the propagation time would imply, due to the commonplace orphaned blocks. For example, if blocks are generated every 15 seconds on average and network propagation time is 20 seconds, then the effective interval between blocks in the main chain in GHOST is at least 55 seconds (this bound is implied by the tightness of [33, Lemma 6]). That is, the payout rate is only about 10 times faster than Bitcoin. In Spacemesh, on the other hand, we can easily support (say) 200 times the Bitcoin payout rate by using a layer width of 200. It should also be noted that GHOST exhibits the nonlinearity of rewards phenomenon (cf. [21, 5]).

Bitcoin-NG The Bitcoin-NG protocol of Eyal et al. [12] seeks to improve upon Bitcoin by offering faster confirmation time and better throughput. The incentive-compatibility aspects of the Bitcoin-NG protocol are not analyzed in [12]—Bitcoin-NG may be susceptible to problems similar to those that Bitcoin has. E.g., when a rational Bitcoin-NG miner sees that the microblocks before the most recent leader L have collected a relatively high amount of fees, she may fork L by extending one of the microblocks that preceded L and leave some of the high fee transactions to next leader who would therefore prefer her fork. Due to the chain structure, the scalability improvements of Bitcoin-NG require sending microblocks of a smaller size as the transaction volume increases, otherwise the competing miners will be discouraged from collecting too many transactions—this becomes impractical for microblock sizes that are too small (the reported experiments in Bitcoin-NG were done with a transaction volume that is at most 5 times greater than that of Bitcoin).

Inclusive Blockchains The work of Lewenberg, Sompolinsky and Zohar [22] presents a cryptocurrency protocol that is based upon a DAG instead of a chain, but unlike Spacemesh it does not provide a mechanism for cooperative sharing of the block rewards. In a sense, [22] is still a blockchain protocol; there is always a main chain, but parties will incorporate non-conflicting transactions from off-chain blocks that do not appear on the main chain. In fact, [22] explicitly states that it does not mitigate selfish mining attacks.

Conflux Li et al. [23] present a DAG protocol that uses the GHOST rule to agree a pivot chain, and then reach consensus on the entire DAG by remove conflicting transactions according to a chronological order—thus it it similar to [22].

OHIE Yu et al. [37] give a protocol that uses k blockchains simultaneously, with rules that dictate a full ordering of the blocks across the k chains. OHIE is similar to a DAG-based protocol in terms of computational and communication complexity, since each node must keep track of all the k chains. Unlike Spacemesh, OHIE is not race-free—when two blocks try to extend one of the k chains at the same height, one of the two blocks will be valid and the other will be orphaned.

FruitChain Pass and Shi [28] present a blockchain protocol where miners who create blocks are not rewarded, but the blocks incorporate "fruits" (similar to "shares" as in p2pool [35]) and miners earn rewards for the fruits that they create. As in Spacemesh, FruitChain attains a protocol that is ϵ-incentive-compatible via a scheme that shares the rewards among those who contributed to the recent ledger history. The analysis of [28] sidesteps the freeloading risk w.r.t. block creators, by using a model in which verification complexity has a negligible cost. Moreover, [28] does improve upon the scalability aspects of Bitcoin, and FruitChain inherits the throughput limitations that chain-based protocols have.

2.4 Leaderless PoW Based Protocols

Blockchain-free Ledger Boyen, Carr and Haines [9] give a cooperative DAG-based protocol, though it only considers security against a rational adversary. While it is not based in leader-election, this protocol is not race-free—conflicts between honest parties may occur due to network delays, in which case some honest parties will not receive a reward.

IOTA The IOTA cryptocurrency [29] was launched in 2017, using a DAG-based protocol. However, IOTA does not have any security proofs; its whitepaper describes only the *typical* case, and considers some potential attacks and countermeasures (most of which are not fully specified). In fact, the main technical problem that Spacemesh addresses—guaranteed convergence towards consensus under a malicious attack—is brushed off as a splitting attack for which no rigorous (or even convincing) solutions are offered. IOTA currently relies on a centralized "Coordinator" node for security [36], and its original design has cryptographic vulnerabilities [26].

SPECTRE Sompolinsky, Lewenberg, and Zohar [32] construct a DAG protocol that guarantees consensus only on blocks that were honestly generated. This relaxed guarantee is enough to construct a cryptocurrency supporting basic monetary transactions, but is insufficient for more advanced uses (such as complex scripts), as they may require consensus on a total ordering of the blocks in order to evaluate even honest transactions. Overcoming the problem of maliciously-generated blocks is the main technical difficulty in constructing and analyzing Spacemesh (if this was not an issue, we could omit the weak-coin protocol, for example). A major design goal of SPECTRE is fast transaction irreversibility. While transactions are confirmed faster in Spacemesh than in Bitcoin, this is not an aspect that Spacemesh excels in.

PHANTOM Sompolinsky and Zohar [34] give a DAG protocol that guarantees eventual consensus on the total block ordering, based on an efficient approximation algorithm of an NP-hard problem. As such, the core of PHANTOM is more complex than our tortoise protocol (an earlier revision of the efficient PHANTOM variant was vulnerable to attacks [23]).

2.5 Proof of Space Based Blockchain Protocols

Chia The Chia network blockchain [11] seeks to emulate a Bitcoin kind of blockchain by replacing each block whose PoW is the first to win the lottery with a block whose PoSpace "quality" is the first to win the lottery. In addition to a VDF, such a process requires an ungrindable PoSpace construction – the particular construction [2] that Chia uses does not have the incremental difficulty property of our PoST mechanism (adjustable difficulty is possible but at the cost of a higher verification complexity). Since blocks are created within minutes in the chain structure, this implies frequent PoSpace initializations and VDF computations that are carried out by individual miners. Consequently, Chia is less green than Spacemesh: individual miners in Chia perform nonstop VDF computations at 100% CPU utilization but only on a single thread, whereas in Spacemesh the sequential work computation is aggregated via PoET servers. Similarly to OHIE, every miner in Chia maintains k chains instead of a single chain – this is done to achieve security that is on par with a PoW based blockchain, it does not improves scalability as only the transactions in the heaviest chain are valid.

Filecoin The Filecoin decentralized backup system [4, 14] uses a blockchain that is constructed via Proof of Replication (PoReplication) and a VDF. PoReplication is a kind of PoSpace that proves that multiple replicas of a single file are being stored, for the purpose of convincing end-users that their backups exist. The overhead of PoReplication is higher than that of other PoSpace mechanisms; the particular PoReplication construction of Filecoin relies on SNARK [17, 19, 7, 3] (with trusted setup) to reduce the complexity. A deficiency of PoReplication is that it does not distinguish between replicas that are stored on the same server and replicas that are stored in multiple locations. Similarly to Spacemesh, Filecoin does not require an ungrindable PoSpace mechanism because each miner submits her PoSpace prior to becoming eligible to create blocks – but frequent initializations are still required as the specific PoReplication mechanism of Filecoin does not support an adjustable difficulty.

2.6 Recent Work on Byzantine Agreement

Most closely related to our hare protocol (cf. section 6) are the works of Abraham et al. [1]. Concretely, our protocol builds on the ideas of [1] to reach agreement, but differs from theirs in that it takes sets of blocks as input and agrees on a common subset of the parties' inputs under some non-triviality conditions. Our hare protocol also resembles [1] in that we also run it in a subcommittee that is randomly selected using a VRF in every round – a (by now) standard approach that was presented earlier in the works of Micali et al. [18, 10]. As opposed these earlier works, our hare protocol has multiple proposers instead of a designated proposer, since the parties need to reach agreement on a set of concurrent blocks as part of the Spacemesh framework. We also describe how to minimize the communication complexity at the protocol level by leveraging the underlying gossip network (section 6.3).

3 Model and Definitions

We use the standard Interactive Turing Machine (ITM) execution model to formalize our protocols. A protocol Π in this formalism specifies how a set of parties (modelled as ITMs) interact with each other. Denoting the security parameter as κ, the execution of Π is directed by an environment $\mathcal{Z}(1^\kappa)$ (also modelled as a PPT ITM) which has full control over the nodes running Π and may corrupt or crash a certain subset of the nodes in an adaptive fashion (see below for more details). Corrupted nodes are under full control of the monolithic adversary $\mathcal{A}$, who sees their internal state and can make them deviate from the protocol arbitrarily, while the remaining honest nodes execute Π according to its description. $\mathcal{Z}$ activates the parties in a round based fashion, providing inputs to them at the beginning of every round and receiving their outputs at the end of that round. We assume that every round coincides with a single time step for simplicity.

Time, Rounds and Layers Our model assumes honest parties have synchronized clocks.[5] We measure time in *rounds*. For simplicity, we will use a round both as the granularity of time measurement and as the bound on communication delay (this is w.l.o.g.; we can always increase the measurement granularity until it holds).

Our protocol also divides time into *layers*. The interval between two layers, **layertime**, is a fixed number of rounds (the length of this interval is a protocol parameter). When the intention is clear, we sometimes use the *layer index* as a point in time—layer i corresponds to the round $r = i \cdot$ **layertime**.

Communication Model Parties in our model are connected by a gossip-like network with the following **network synchrony** properties:

- If an honest node sends a message m at time t, then every honest party receives m time $t + \delta$ (or earlier). By comparison, the adversary $\mathcal{A}$ can send a message m to only an arbitrary *subset* of honest parties.
- If an honest node receives a message m at time t, then every honest party receives m at time $t + \delta$ (or earlier).

Adversary Model At any time during the protocol execution, $\mathcal{Z}$ can spawn new nodes, each of which is either corrupt or honest. As already outlined above $\mathcal{Z}$ can also adaptively corrupt nodes at any given point in time by sending them the command `corrupt`, upon which the adversary learns the corrupted party's internal state and henceforth controls that party, after Δ time steps have elapsed. This models that corruptions take some time to become active. On the other hand, the adversary can issue `sleep` and `wake` commands to honest parties, which take effect immediately. More precisely, upon receiving a command `sleep`, a party stops executing the protocol Π and falls asleep, i.e., it stops sending or receiving protocol messages and stops any computation related to executing Π. Upon receiving a `wake` command, a sleeping party rejoins the protocol immediately and all messages that it ought to have received to it while it was asleep, are delivered upon waking.

Resource Model We consider two main resources for this work: Storage and computation (hash) power. We model storage by giving every ITM a storage tape of bounded length. At the beginning of every round, all the tapes apart from the storage tapes are reset to their initial state (thus, a node can use a work tape of arbitrary length, but the data it transfers from one round to the next must fit in the bounded storage length). We assume the cost of storage is linear in the size of the storage and in the time it is used (think of this paying some cost s to "rent" one unit of storage for one round)

To model computation resources, we assume a public hash oracle H (modeled as a random oracle) and measure the computation cost of a party by the number of oracle queries it makes). For simplicity,

[5] This is done to simplify the writeup, however, the protocol can handle small differences in local clocks, by including them in the communication delay.

we will normalize all costs by the cost of one "computation unit"; thus, one oracle query always costs 1, while storage costs may vary over time.

$(1-q)$-Honest Majority. One of our basic security assumptions is that the adversary's total resources are bounded. For any protocol round r, let the total cost associated with all hash queries and rented units of storage of active parties during this round be c_r. The $(1-q)$-Honest Majority assumption is that total cost of the resources expended by the adversary is at most $q \cdot c_r$.

Blockchain notations We use the notation `chain` to denote our linearized blockmesh. We write `chain`$[:-L]$ to the entire chain except for the trailing L blocks; `chain`$[:L]$ denotes the entire chain up to block L; `chain`$[-L:]$ denotes the trailing L blocks of chain. We say that `chain` is 'an honest chain in `view`', iff `chain` is some honest (and online) node?s output to the environment $\mathcal{Z}$ in some round in view. We use the notation `chain`^{t}(`view`) to denote node a chain in round t in `view`.

Mining a Block We say that a node mines a block on chain `chain` if there is a set txs such that $txs \subset$ `chain`$[-1:]$ and moreover $\mathcal{Z}$ input txs to some honest node when its last output to $\mathcal{Z}$ contains `chain`.

3.1 Security Properties

In this subsection we introduce the security properties that we consider in this work. We again use the definitions of [27]. In the following, consider a blockmesh protocol $\Pi_{blockmesh}$ and say that it is *secure* if the following properties hold with overwhelming probability for `view` $\leftarrow \mathsf{EXEC}^{\Pi_{blockmesh}}(\mathcal{A}, \mathcal{Z}, \kappa)$ w.r.t. some $(\mathcal{A}, \mathcal{Z})$:

Chain Growth. If in round r some honest chain is of length ℓ, then in round $r+\delta$, all honest chains must be of length at least ℓ.

(T, μ, L)-chain quality. `view` satisfies (T, μ, L)-chain quality iff the following holds: Let `chain` denote any honest chain in `view`. Then for any T consecutive blocks in `chain`, denoted as `chain`$[j+1, ..j+T]$ for some $j > 0$, then more than a μ-fraction of those blocks were mined by some honest node on the prefix chain `chain`$[:i-L], i \geq j+1$.

T-consistency. `view` satisfies T-consistency iff the following hold: for any two honest chains `chain`^{r} and `chain`^{t} in round r and $t \geq r$ respectively, it holds that `chain`$^r[:-T]$ is a prefix of `chain`^{t}. Since `chain`^{r}and `chain`^{t} can belong to the same node, the above definition also implies future self consistency of any chain, except for the trailing T blocks.

4 The Protocol

The purpose of the Spacemesh consensus protocol is to achieve a permissionless consensus on an "append-only ledger". The ledger contains a list of *transactions*; for the purposes of the protocol, transactions are opaque strings—the protocol doesn't interpret them in any way (for optimization purposes, a cryptocurrency built on top of the ledger can add additional restrictions, in order to prevent clearly invalid transactions from entering the ledger in the first place, but we ignore that here).

More formally, we achieve the properties of an abstract blockchain protocol, as defined by Pass et al. [27].

4.1 From Blockmesh to Ledger

We implement the ledger by implementing consensus on a blockmesh—a layered DAG in which each node is a *block*, each block is a member of an indexed *layer*, and edges only point "backwards": i.e., a block in layer i can have an edge to a block in layer j only if $j < i$.

Block IDs We assume each block has a unique ID that can be computed from the contents of the block. W.l.o.g., the ID can be the entire contents, but in practice we use a collision-resistant hash of the contents. For a block B, we denote id_B the ID of block B.

Total Order on Blocks We use the following rules to define a full ordering between blocks. Let A be a block in layer i and B a block in layer j. Then the order between A and B is the lexicographic order between the tuples (i, id_A) and (j, id_B), where the first element is most significant (i.e., order first by layer, then by ID).

Total Order on Transactions Each block contains an ordered list of transactions. These lists are not mutually exclusive—the same transaction can appear in multiple blocks. To get a full order on transactions from the blockmesh, we consider only the first block (according to the total block order) in which each transaction appears. If transactions s and t appear in different blocks, they are ordered according to the block ordering. If they appear in the same block, they are ordered according to their indices in the block.

From Total Order to Consensus on Complex State The layer above "consensus" in the cryptocurrency protocol stack is the "consensus computer": in this layer the participants agree on the state of a virtual computer and its continuing evolution. Once users agree on a total order of *transactions*, the agreement on the current state of the virtual computer is immediate.

Starting from genesis (where the virtual computer has, by definition, an agreed initial state), every transaction modifies the state deterministically. To compute the current state, users execute all transactions, in their agreed order.

4.2 Non-Interactive Proof of Space-Time (NIPoST)

A PoST, formally defined in [24], is a two phase protocol. In the first phase, the prover *initializes* a given amount of storage, with respect to a specific ID (seed). The initialization process involves filling the storage with pseudorandom data; this data is the result of a computationally-expensive (but easy to verify) operation. In the second, execution, phase, the prover receives a challenge, and proves that the data is still stored (or was recomputed). The execution phase can be repeated arbitrarily many times without repeating initialization; thus despite the initialization essentially serving as a proof-of-work, the amortized computational complexity can be made arbitrarily small.

Unlike a PoST, a NIPoST has only a single phase. Given an id, a space parameter S, a duration D and a challenge ch, a NIPoST is a single message that convinces a verifier that (1) the prover expended $S \cdot D$ space-time *after* learning the challenge ch. (2) the prover did not know the NIPoST result until D time after the prover learned ch.

Measuring Duration The duration of a NIPoST isn't measured in clock time (since that cannot be publicly verified). Instead, the NIPoST uses *sequential work* as a proxy for time. That is, we measure duration as a number of sequential CPU operations. Which operations depends on the construction, e.g., a random-oracle based construction would measure hash invocations, while an algebraic construction could use sequential multiplications. We abstract this, referring to the base sequential operation as a *sequential work tick*.

NIPoST API More formally, a NIPoST scheme consists of three algorithms: InitializeNIPoST(id, S), GenerateNIPoST($id, S, D, ch, \mathrm{com}, \sigma$) and VerifyNIPoST($id$: seed ID, S: space, D duration, ch: challenge, π: proof) (see fig. 4.3).

We define NIPoST soundness using the NIPoST R-security game, between an adversary $\mathcal{A}$ and a challenger:

1. $\mathcal{A}$ sends n and a set of n tuples to the challenger: $(id_1, S_1), \ldots, (id_n, S_n)$

2. The challenger uniformly chooses a challenge $(\mathrm{ch}_1, \ldots, \mathrm{ch}_n)$

3. The adversary sends a set of $n' \leq n$ proof tuples $(D_1, \pi_1), \ldots, (D_{n'}, \pi_{n'})$.

We say the $\mathcal{A}$ wins the R-security game if

$$\left(\max_{i \in [n']} D_i\right) \cdot \sum_{i=1}^{n'} S_i \geq R$$

(i.e., the total space-time resource the adversary claims to have used is at least R) and for all $i \in [n']$, VerifyNIPoST$(id_i, S_i, D_i, \mathrm{ch}_i, \pi_i) = 1$.

Definition 4.1 (NIPoST Soundness)**.** We say a NIPoST protocol is α-sound if, for every PPT adversary $\mathcal{A}$ with expected number of sequential work ticks D^* and total storage S^*, the probability that $\mathcal{A}$ wins the $\alpha D^* \cdot S^*$-security game is negligible.

4.2.1 Constructing a NIPoST from a PoST and a PoET

The given a PoST and a PoET, the construction of a NIPoST is straightforward: the PoST commitment will serve as the PoET challenge, while the PoET will serve as the PoST challenge. The proof itself consists of the PoST commitment, the PoET proof and the PoST proof. We need the PoST to have sigma-protocol form—the initialization must consist of the prover sending a single "commitment" message, and the execution phase of the verifier sending a random challenge and the prover responding with a single proof message (the API for a sigma-PoST is summarized in fig. 4.1, and for the PoET in fig. 4.2).

Figure 4.1: Sigma-PoST API

```
1:  function InitializePoST(id: seed ID, S: space )
2:      // Compute com: the PoST commitment and σ: the prover state (storage)
3:      return (com, σ)
4:  end function
5:  function VerifyPoSTInit(id: seed ID, S: space, com)
6:      return true or false
7:  end function
8:  function ExecutePoST(id: seed ID, σ : prover state, ch: challenge )
9:      return π  // proof
10: end function
11: function VerifyPoST(id: seed ID, com, ch: challenge, π: proof)
12:     return true or false
13: end function
```

Figure 4.2: PoET API

```
1: function PoET(ch: challenge, D: duration)
2:     return π  // Proof
3: end function
4: function VerifyPoET(ch: challenge, D: duration, pi: proof)
5:     return true or false
6: end function
```

Figure 4.3: NIPoST Construction

function InitializeNIPoST(id: seed ID, S: space)
  $\sigma = (\text{com}, \sigma_1) \leftarrow$ InitializePoST(id, S)
  // com: PoST commitment; σ_1: PoST state (storage)
  return σ
end function
function GenerateNIPoST(id: seed ID, S: space, D duration, ch: challenge, σ)
  $(\text{com}, \sigma_1) \leftarrow \sigma$
  $\pi_1 \leftarrow \text{PoET}(H(id \| \text{com} \| ch))$
  $\pi_2 \leftarrow$ ExecutePoST$(id, \sigma_1, H(\pi_1))$
  return $\pi = (\text{com}, \pi_1, \pi_2)$
end function
function VerifyNIPoST(id: seed ID, S: space, D duration, ch: challenge, π: proof)
  Denote $(\text{com}, \pi_1, \pi_2) \leftarrow \pi$
  if VerifyPoSTInit$(id, S, \text{com}) = false$ then **return false**
  if VerifyPoET$(H(id \| \text{com} \| ch), \pi_1) = false$ then **return false**
  if VerifyPoST$(id, \text{com}, H(\pi_1), \pi_2) = false$ then **return false**
  return true
end function

Intuitively, an adversary cannot guess the commitment value without first generating the PoST data (otherwise it could either break the binding property of the commitment or the initialization work property of the PoST), and cannot guess the PoET outcome before the elapsed duration since learning the PoST commitment (otherwise it could violate PoET security).

4.3 Coin Flips and Beacons

The Spacemesh protocol uses three, slightly different, types of "random beacons". Loosely speaking, a beacon is a "random" string on which all honest parties agree at time t, such that (1) the adversary cannot guess the value of the beacon before time t, and (2) cannot influence the value of the beacon "too much" (we give more formal definitions below). Constructing a beacon is not a trivial task—in particular it implies a form of weak coin flipping.

Mitigating this difficulty somewhat, our protocol doesn't require the beacons to be publicly verifiable. This means it's enough that honest parties agree on the beacon value; we don't care if the adversary can "lie" about it. Publicly-verifiable beacons are much more difficult to achieve; this relaxation makes implementing the beacons simpler and more robust.

We give a parameterized, formal definition for the properties we require from a beacon, and then explain the parameters for each of our three beacons.

Definition 4.2 ($n, p, c, \varepsilon, \delta, t$-Beacon)**.** We say a functionality $\mathcal{F}$ that outputs a string $s \in \{0,1\}^n$ is a $n, p, c, \varepsilon, \delta, t$-beacon if it satisfies:

1. p**-consensus:** At time t, all honest parties agree on s with probability at least p.
2. ε**-unpredictability:** The probability that the adversary correctly guesses s at any time $t' < t - \delta$ is less than ε.
3. c**-bounded influence:** For any efficiently-computable function f and any value y,

$$\Pr[f(s) = y] < c \cdot \Pr[f(U_n) = y]$$

(where the probability is over the randomness of the adversary and the functionality, and U is uniformly distributed over $\{0,1\}^n$)

- **k-layer challenge**: The challenge for layer i, denoted ch_i, is the output of a $k, 1-\mathrm{negl}(k), 1+\mathrm{negl}(k), \mathrm{negl}(k), \mathrm{k}, i$-beacon. That is, the probability that the adversary can output ch_i) before the start of layer $i-\mathrm{k}$ is negligible. On the other hand, at the start of layer i, all honest parties must agree on the value of ch_i except with negligible probability. (This challenge is used to determine eligibility in the hare protocol committees; since it is not used in the tortoise protocol, self-healing does not depend on the validity of this beacon, so implementations can use the fact that honest parties agree on history at layer $i-\mathrm{k}$ with high probability).

- **p-coin flip**. We denote coin_i the result of the coin flip for layer i. The coin flip is the output of a $1, 2p, 2-2p, 1-p, 1, i$-beacon: No adversary can predict coin_i with probability more than $1-p$ before the start of layer $i-1$. Moreover, with probability at least p, all honest parties agree that $\mathrm{coin}_i = 0$ at the start of layer i, and with probability at least p all honest parties agree that $\mathrm{coin}_i = 1$ at the start of layer i. (The coin flip is used to achieve self-healing while under adversarial attack. Note that this is a much weaker requirement than the layer challenge, since with probability $1-2p$ the adversary can control the coin value *for each honest party separately*)

- **epoch beacon**: The beacon for epoch z is denoted ech_z (see section 4.4.1 for an explanation of epochs). It is the output of a $k, 1+\mathrm{negl}(k), \mathrm{negl}(k), \Delta_{\mathrm{epoch}}, z\cdot\Delta_{\mathrm{epoch}}$-beacon. That is: honest parties agree on ech_z at the start of epoch z, no adversary can predict ech_z before the start of epoch $z-1$ and it behaves "almost" like a uniform string of length k. There are two important differences between the epoch beacon and the layer beacon: on the one hand, the epoch beacon cannot depend on the hare protocol working, since we use it to bootstrap the hare protocol when in self-healing mode. On the other hand, we only need one epoch beacon per epoch, so we can use a slower implementation to achieve it.

Implementing the beacons All three beacons can be "perfectly" implemented using a Verifiable Delay Function (unique PoSW, cf. [8]): the VDF can be iterated sequentially, starting with the genesis block as the initial input, and its outputs at each layer used as the beacon value. Since it's a *delay function*, the adversary cannot learn the value of the beacon before the honest parties, and since it's deterministic the adversary cannot influence the output in any way.

However, all existing VDFs rely on algebraic complexity and implementation assumptions that are not very well studied (compared to, for example, sequential complexity of SHA256). Below, we sketch possible implementations that do not rely on a VDF (we defer a detailed description and proofs to the full version):

- **k-layer challenge**: Since implementations of this beacon can assume honest parties agree on "old-enough" history, we can use a hash of the (agreed) history as the challenge. (If this assumption is violated—for example, if there was an extremely long network split—then the hare protocol will stop working, but the tortoise self-healing will eventually ensure honest parties are in consensus on history again.)

- **p-coin flip**: For the layer-i coin flip, we can sort the layer-i VRF outputs in the layer-i eligibility proofs, and take the hash of the first one according to this order. If this turns out to be an honest identity, all honest blocks will have received it, so will agree on the result (and it is also unpredictable to the adversary). On the other hand, since the adversary is limited in the number of valid blocks it can generate at layer i, the probability that an adversarial block is chosen is approximately bounded by q.

- **low-influence epoch beacon**: We can use a combination of the coin-flip and a modified version of the tortoise protocol itself to implement this beacon. The basic idea is to sample identities in epoch $z-1$ using the VRF output (e.g., check if the VRF output passed a "difficulty threshold"),

such that on average $3k$ identities are sampled. The parties then XOR the VRF outputs from the selected identities to compose the beacon value. Since the adversary can't control the value of the unique signature, only whether or not it appears, it has one bit of influence for every selected ATX. If $q < 1/3$, this means on average it has less than k bits of influence. This isn't quite sufficient, since honest parties might disagree on which IDs were sampled (e.g., if the adversary delays sending a "successful" identity proof). To overcome this, the parties use epoch z to reach consensus about which ATXs actually passed the threshold, by voting for them using the tortoise protocol's self-healing mechanism (we use a modified version of the protocol that ignores the votes from "heavy" layers, in order make grinding on the previous epoch's beacon ineffective).

4.4 Active IDs

Miners "activate" their IDs by publishing *Activation Transactions* (ATXs). Each activation transaction is a proof that the miner expended some space-time resource, making the miner potentially eligible to generate blocks (or provide other services to the network, such as participate in Hare committees).

4.4.1 Epochs, Activation and Maturation

We divide time into *epochs*, where each epoch is a fixed number of layers (denoted Δ_{epoch}). An epoch should be long enough so that each active miner can generate at least one block in every epoch (e.g., if there are 100000 active miners, and 200 blocks per layer, each epoch should be at least 500 layers long).

Generating a new identity requires a miner to publish an ATXs for that identity in k separate epochs (this makes it more expensive to "grind" on identities, and allows us to use beacons with weaker properties). The first k ATXs are considered *immature*; every succeeding ATX for the same identity is *mature*.

A valid, mature ATX activates the associated identity for a single epoch. A node ID is considered "active" starting at the first epoch boundary following the publication of the ATX (e.g., if an epoch is $\Delta_{\text{epoch}} = 1024$ layers, and a mature ATX was published in layer 1700, the ID becomes active at layer 2048, and remains active until layer 3072).

4.4.2 ATX Contents

An ATX is a "free-standing" transaction—it does not have to be included in a block to be valid. Each ATX is a tuple containing the following elements:

1. **Node ID**: a tuple $(vk_{id}, \text{vrf}_{id})$, where vk_{id} is a verification key for a signature scheme and vrf_{id} is the verification key for a VRF[6].
2. **Sequence number** s: a sequence number (ATXs for a given Node ID must be published in sequence).
3. **Previous ATX**: the ID of the ATX with the same Node ID and sequence number $s - 1$. (if $s = 0$ then this field is empty).
4. **Layer Index** i: the index of the layer at which this ATX becomes valid.
5. **Positioning ATX**: the ID of an ATX (not necessarily with the same Node ID) with a layer index $i - \Delta_{\text{epoch}}$. Miners should always choose the ATX with the highest end-tick at layer $i - \Delta_{\text{epoch}}$.
6. **Start tick** t: The "sequential work tick" at which this ATX starts. This is equal to the *end tick* of the positioning ATX.
7. **End tick** t': the sequential work tick at which this ATX ends.

[6]These can be combined into a single key if the signature scheme has unique signatures, and can thus be used as a VRF

8. **NIPoST**: a NIPoST proof using the Node ID as the ID and a hash of all the previous fields as the challenge. The NIPoST duration must be $t' - t$.

9. **Active set size** d: the number of "active" ATXs seen by the node creating this ATX at the time it is published (note that d is *not* an input to the NIPoST).

10. **View pointers**: pointers (IDs) of blocks whose combined visible mesh (see section 4.5) includes d active ATXs. (This will usually be a single block, but can include explicit pointers to ATXs not in any block)

11. **Signature**: A signature of the ATX ID (the hash of the ATX excluding the signature) using the private key corresponding to vk_{id}.

(For clarity, we describe each element separately above, but for efficiency purposes they may be combined or deduced from other elements—for example, the starting tick is the ending tick of the positioning ATX, so doesn't need to be explicitly included, while the ending tick can be computed from the starting tick and the NIPoST difficulty)

4.4.3 ATX validity

An ATX is (syntactically) *valid* if it satisfies the following conditions:

- The signature verifies.
- No other ATX was published with the same ID and sequence number.
- If $s \neq 0$ then the Previous ATX is valid and its sequence number is $s - 1$.

  Together with the previous condition, this ensures that each ATX requires its own resource expenditure; without the sequence numbers, it would be possible for an adversary run several NIPoSTs *in parallel* for a given storage space, thus generating several ATXs but expending space-time only once.

  Note that the ATX is *mature* iff $s \geq k$.
- The Positioning ATX is valid and its end tick is t.
- The NIPoST verifies with Node ID as its ID, D as its duration and a hash of fields 1 to 7 as its challenge.

  (This ensures that the ATX can only have been generated D ticks after the Positioning and Previous ATXs)
- The layer index i' of the positioning ATX satisfies $i - i' < D/\textbf{layertime}$, where D is the NIPoST duration.

  Together with the previous conditions, this ensures that an adversary can't "shift" its resources into the future; without a Positioning ATX, the adversary could generate valid ATXs with layer numbers in the future. This would allow the adversary to concentrate resources in a short period by shifting multiple IDs to the same future time (e.g., it generates ATXs for different identities one after another—reusing the same space—but publishes all of them in a single future epoch.)
- The ATX's view contains d active IDs.

  (This ensures that the adversary can't inflate d; doing so would allow it to reduce the number of blocks it generates per epoch, concentrating its voting weight on fewer layers)

ATX Weight Each ATX has a "weight", which is used in both the reward calculation and in the tortoise contextual validity algorithm. See section 4.5.2 for more details.

ATX Communication Complexity Having *every* miner publish an ATX at a constant rate would cause the communication complexity of the protocol to grow linearly with the number of miners (as opposed to Bitcoin, for example, where the rate is independent of the number of miners, since the PoW difficulty is adjusted to keep the block rate constant).

We utilize a similar "difficulty" adjustment to keep communication rate constant: by increasing epoch length as the number of miners grows (and correspondingly the PoST initialization difficulty, to ensure storage remains the rational option), we increase the interval between activations to make the overall ATX transaction rate constant.

To make this writeup clearer, and focus on the more difficult problems the protocol solves, we will defer the description of this difficulty adjustment mechanism to the full version. We note that for practical purposes, an epoch length of 2 weeks (4032 layers) would suffice for over 800000 miners; in the worst case, with every miner contributing only a single block per epoch, the communication overhead is equivalent to adding a single (albeit fairly large) transaction to each block.

4.5 Blocks

A block is a tuple containing the following items:

- **Layer counter** i: The layer in which this block claims to be included in.
- **Tick** t: The work tick this block claims is associated with layer i. This is computed from the block's ATX in the following way: if the ATX is at layer j, has a start-tick s' and an end-tick t', then
$$t \leftarrow t' + (i - j) \cdot (t' - s')/\Delta_{\text{epoch}}$$
(that is, use the end-tick corresponding to the ATX, and compute the number of ticks per layer using the ATX duration).
- **Node ID** id: ID of generating node.
- **Eligibility Proof** π: A proof that the id is eligible to create a block in layer i (see section 4.5.1)
- **Visible Mesh**: IDs of all visible (syntactically-valid) blocks in previous layers. It's enough to explicitly include only the "roots" of the blockDAG (blocks with in-degree 0); the rest of the visible mesh can be deduced by running a DFS starting with the root blocks.
- **Hare votes**: IDs of all blocks that are accepted by Hare protocol (i.e., "good", recent layer blocks). A recent block whose ID is not in this list is considered to have been rejected by the Hare protocol. (For optimization purposes, we store Hare votes with one layer of indirection: the block contains the ID of a *pattern*, where a pattern is a subset of blocks in a particular layer.)
- **Common coin**: the common coin bit for this block layer (according to the node that generated the block).
- **Transactions**: IDs of all transactions included in the block.

We note that the items on this list do not have to appear *explicitly* in the encoding of a block—for example, IDs of the Hare votes can be encoded as paths to those blocks in the visible mesh.

4.5.1 Block Eligibility

A proof of eligibility for a node with ID id to create a block in layer i consists of:

- Most recent ATX for $id = (vk_{id}, \text{vrf}_{id})$.
- An index $j \in \left[0, \left\lfloor \Delta_{\text{epoch}} \cdot T_{\text{ave}}/d \right\rfloor\right)$.

- The value of the beacon for this epoch ech_z, where $z = \left\lfloor i/\Delta_{\text{epoch}} \right\rfloor$ is the epoch number. The beacon value must be identical for all blocks with the same ID in a given epoch.
- The output $i^* \leftarrow \text{vrf}_{id}(\text{ech}_z \| z \| j)$

The eligibility verification function appears in fig. 4.4. Note that the bound on the index j ensures that for each epoch in which an ID is active, the ID will be eligible to generate exactly $\left\lfloor \Delta_{\text{epoch}} \cdot T_{\text{ave}}/d \right\rfloor$ blocks; if d is the actual size of the active set, this means that the total number of blocks in an epoch will be $\Delta_{\text{epoch}} \cdot T_{\text{ave}}$, hence the expected number of blocks in a layer is T_{ave}. To determine the layer (within the epoch) in which the j^{th} block is generated, we parse the VRF output on $\text{ech}_z \| z \| j$ as a pseudorandom index in the range $[0, \Delta_{\text{epoch}})$.

Figure 4.4: Block Eligibility Verification

```
1:  function IsEligible(X: block)
2:      Let id = (vk, vrf) ≐ id_X, i ≐ layer_X
3:      Let π = (ATX, j, i*) be the eligibility proof.
4:      Let z ← ⌊i/Δ_epoch⌋ be the block epoch.
5:      Let z' ← ⌈layer_ATX/Δ_epoch⌉ be the ATX epoch.
6:      if ATX is valid, id_ATX = id, z = z', i* = vrf_id(ech_z||z||j) and i* ≡ i (mod Δ_epoch) then
7:          return true
8:      else
9:          return false
10:     end if
11: end function
```

4.5.2 Voting Weight

The core of our tortoise protocol is vote counting. The "voters" in the protocol are the miners. Each unit of space-time expenditure by a miner—attested to by an ATX—gives the miner a single vote. For simplicity, we assume space is always a single unit, so the weight of an ATX is the NIPoST duration (this duration is recorded in ticks, not wall-clock time—so as the PoET difficulty increases, the weight of new ATXs increases proportionally).

The votes themselves are recorded in the blocks. Each ATX "commits" to d, which determines the number of blocks it is eligible to generate in the next epoch: $\left\lfloor \Delta_{\text{epoch}} \cdot T_{\text{ave}}/d \right\rfloor$. Thus, the *voting weight* of each block is $w/\left\lfloor \Delta_{\text{epoch}} \cdot T_{\text{ave}}/d \right\rfloor$, where w is the weight of its ATX.

If a node "double votes" by creating two blocks with the same ID in the same layer, or two ATXs with the same sequence number, all blocks with that ID have their weight set to 0. (Although this does allow an adversary to change "historic" votes, the protocol ensures that doing so can't change any honest node's opinion about the contextual validity of nodes about which it is already confident.)

Neutral and postponed votes The total weight of the adversary's votes is always one vote per ATX. From our assumption on the adversary's resource bounds, it follows that the total fraction of ATXs generated by the adversary at any given time is at most q, hence its fraction of the votes is also bounded by q.

However, this is only true as a global statement. For specific layers, the adversary might have a larger fraction of votes. If the adversary manages to "concentrate" voting power in a small number of layers, it can potentially "overwhelm" the honest votes and cause problems.

The adversary can concentrate its votes in two ways. The first is by "time-shifting" entire ATXs. For example, suppose it can use epochs 1–100 to generate ATXs that ostensibly belong to layers 101–200

with id_1. It can then erase id_1 and reuse the storage to create id_2, and generate ATXs for id_2 in layers 101–200. This means it has twice its actual weight in these layers. This attack can be used to shift an arbitrary amount of weight to some future date.

The second concentration attack involves shifting weight in a specific epoch. While the total weight in an epoch is fixed, the adversary can influence in which layers it is eligible to mine blocks.

To prevent both types of concentration attacks, we add the following exception to the general voting rule. In the view of party P, a block B votes (explicitly or implicitly) for every block X such that $\mathrm{layer}_X < \mathrm{layer}_B$, with the following exceptions:

1. If the work tick of P's ATX in B's epoch is greater than B's work tick (i.e., P sees B as using a "slow" PoET), B always votes neutral (with 0 weight).

2. If B is less than ℓ epochs after X, and the epoch beacon that B considers valid for X's epoch differs from the actual epoch-beacon used by X, then B considers X to have weight 0 when counting votes. That is, any honest party that sees a beacon different from X considers all of X's votes *postponed* for ℓ epochs. (ℓ is a tunable parameter; see section 4.6.2)

Condition 1 prevents an adversary from concentrating its voting weight by using a sequence of low-difficulty ATXs to reach a future layer. Since the "fake" ATX in the future will have a start-tick in the present, the "fake-future" blocks won't be able to vote for actual future blocks.

Condition 2 prevents an adversary from concentrating its voting weight by choosing a different beacon. The only elements under the adversary's control in determining the layers in which blocks are mined are the id, ech_z and d—and the latter only allows the adversary to generate blocks at additional locations by reducing the weight of each block. If the adversary uses the honest beacon, which it cannot predict or control (since it's a low-influence beacon), choosing different IDs won't help—the adversary must choose the ID before it learns the outcome of the beacon. Thus, the only remaining option is to modify the beacon. However, by doing so condition 2 prevents the extra weight from affecting honest parties' decisions until the end of the epoch (after which only the total weight for that epoch matters).

In section 5.2, we prove that these measures suffice to ensure that adversarial voting weight is not "too concentrated" in any small set of layers.

4.6 Protocol Parameters

The Spacemesh consensus protocol can be tuned in various ways, and also depends on some external (empirically-set) values.

4.6.1 External Parameters

- δ**, bound on network latency**: for every two honest parties A and B, if A receives/sends a message at time t according to A's internal clock, B will receive the message by time $t + \delta$ according to B's internal clock. (In this writeup we fix measure time in rounds, where each round is of length δ; based on measurements of the Bitcoin gossip network, rounds of 30 seconds seem reasonable)

- q**, fraction of resources controlled by adversary**

- **storage to CPU cost ratio**: This, together with the epoch length, gives a lower bound on the required initialization cost for the PoST (we require that initialization is more expensive than storing data for an entire epoch). If we assume electricity costs of only \$0.1/kWh (lower than the average US electricity cost), assume a miner's computer uses 500W when initializing at 100% load (gaming computers usually have higher power requirements under full GPU load) and require users to initialize 200GB of storage for the equivalent of two days, the cost of initialization counting electricity alone is \$2.40. Even at Amazon's "standard" storage tier of \$0.021/GB/month, this is equivalent to more than two weeks of storage (using the extra space on a disk you already have is much cheaper of course).

4.6.2 Tunable Parameters

Tunable parameters can be set by the protocol designers (subject to constraints relating them and the external parameters)

- **Layer interval (layertime)**: time between increments of layer counter for honest parties. The layer interval at least 9 rounds, to allow time for the Hare protocol to complete. (To simplify analysis, it can be helpful to think of the layer interval as long enough for the hare protocol to complete within a single layer w.h.p.)
- **Hare Distance (hdist)**: maximum number of recent layers whose validity is determined by the Hare protocol; e.g., when hdist = 2 the parties can vote explicitly on the validity of layers $t - 2$ and $t - 1$ at time t, using the results of the hare protocol. (If the layer interval is long enough for the hare protocol to complete w.h.p. within a single layer, we can set hdist = 1).
- **Epoch length (Δ_{epoch})**: see section 4.4.1.
- **Average layer width (T_{ave})**: this is the average number of blocks per layer. We sometimes use $n \doteq T_{\text{ave}} \cdot \Delta_{\text{epoch}}$ to denote the total number of blocks per epoch.
- **Bad beacon postponement distance (ℓ)**: this is the number of previous epochs for which we consider a block as having weight zero if its epoch beacon differs from ours (see section 4.5.2).
- **NIPoST initialization difficulty**: Used to offset an increasing cost of spacetime (for simplification, in this writeup we will treat this as constant in our analysis).
- **Confidence thresholds: (θ_G, θ_L):** thresholds (expressed as a fraction of the current expected layer weight) used in the tortoise protocol to determine confidence in block validity and whether to use majority or coin flip for validity (see fig. 4.7).

4.7 Syntactic Validity

In order for a block B to be syntactically valid in layer i, it must satisfy the following properties:

- **Active Node ID**: The Node ID embedded in B must be *active*. This can be verified by checking the Activation TX pointed to by the block.
- **Node Eligibility**: The Node ID embedded in B must be *eligible* to generate a block in layer i (see section 4.5.1 for details).
- **Recursive Syntactic Validity**: Every block in B's visible mesh must have been received and be syntactically valid. (Note that if B references a block that hasn't yet been received, B is considered syntactically invalid. Thus, honest parties may briefly disagree on the syntactic validity of B, but our network synchrony assumption guarantees that if any honest party considers B syntactically valid then all honest parties will consider B syntactically valid in the next round.)
- **Transaction Syntactic Validity (optional)**: Every transaction included in the block is syntactically valid. Syntactic validity of transactions can depend only on its contents, and not on the order relation to other transactions (this is just an optimization; it is also ok to depend on the order of sufficiently old transactions, for which consensus and irreversibility already hold with high probability).

The high-level pseudo-code for determining syntactic validity appears in fig. 4.5.

Figure 4.5: Syntactic Validity (High-Level)

```
1:  function IsSyntacticallyValid(B: block)
2:      if B is genesis block then
3:          return true
4:      end if
5:      if not IsEligible(B) then
6:          return false
7:      end if
8:      for B′ ∈ vis do
9:          if not IsSyntacticallyValid(B′) then
10:             return false
11:         end if
12:     end for
13: end function
```

4.8 Contextual Validity

4.8.1 Voting Rules

In order to be contextually valid at time t, block B in layer $i < t$ must be syntactically valid and satisfy two properties:

- **Majority rule**: evaluating this rule depends on the value of $t - i$

Case 1: If $t - i \leq$ hdist, the contextual validity is determined by executing the *hare* validity function.

Case 2: If $t - i >$ hdist, the contextual validity is determined by executing the *tortoise* validity function.

- **Unique ID rule**: Every node ID can generate at most one block per layer. If two blocks in the same layer have the same node ID and both are considered valid according to the majority rule, only the block with the lowest ID (lexicographically) is considered valid.[7]

Note that a block can be contextually valid even though it was received late (otherwise honest nodes might never reach consensus on blocks that appear late to a minority of the honest nodes).

The high-level pseudo-code for determining contextual validity appears in fig. 4.6.

4.9 Simple Tortoise: Complete Voting Transcripts

We first describe a simplified (but communication-inefficient) version of the Tortoise protocol. In this version of the protocol, each block, in addition to the items specified in section 4.5, includes a "vote" (accept/reject) for *every* block in the visible mesh. (In section 4.10, we will show how to optimize the protocol so that this explicit list can be omitted.)

In a nutshell, to determine the validity of a block B, the tortoise validity function counts votes on B's validity from every block in the succeeding layers; if there's a significant majority in either direction (i.e., *for* or *against*), the majority rules (in this case, we say the result is "confident"). Otherwise, considers the vote margin (difference between the total weights of positive and negative votes), and compares it to a smaller threshold. Again, if there's a significant majority, the majority rules (in this case, the result is "tentative"). Otherwise, the node uses the value of the common coin to decide on validity.

An important point is that syntactically-valid blocks can get a vote even if they're not contextually valid. This prevents recursive loops in computing validity. A more subtle issue is that each node ID is

[7] Note that this can only happen if our underlying network connectivity or honest-majority assumptions have been violated.

Figure 4.6: Contextual Validity (High-Level)

```
1: function IsMajorityValid(B: block, t: current time, Ctx: context)
2:     if not IsSyntacticallyValid(B) then
3:         return false
4:     end if
5:     Let (id, i, π, vis) ← B
6:     if t − i < hdist then
7:         if not IsHareValid(B, t, Ctx) then
8:             return false
9:         end if
10:    else
11:        if not IsTortoiseValid(B, t, Ctx) then
12:            return false
13:        end if
14:    end if
15:    return true
16: end function
17: function IsContextuallyValid(B: block, t: current time, Ctx: context)
18:    if not IsMajorityValid(B, t, Ctx) then
19:        return false  // Failed majority rule
20:    end if
21:    if ∃B′ ∈ Ctx, B′ = (id, i, π′, vis′) such that IsMajorityValid(B′, t, Ctx) and ID(B′) <
   ID(B) then
22:        return false  // Failed uniqueness rule
23:    end if
24:    return true
25: end function
```

only allowed one vote per layer. Double-voting does not invalidate *blocks*, but does invalidate the votes themselves (see section 4.9.1 for details).

The complete validity function appears in fig. 4.7.

Figure 4.7: Tortoise Validity

```
1:  function IsTortoiseValid(B: block, t: current time, Ctx: context)
2:      Denote i the layer of block B.
3:      Let (v⁺, v⁻) ← CountVotes(B, i + 1, Ctx)  // Global vote count
4:      if v⁺ > θ_G · T_ave · (t − i) then return true  // Confidently valid
5:      else if v⁻ > θ_G · T_ave · (t − i) then return false  // Confidently invalid
6:      else // Global margin too small;
7:          if v⁺ − v⁻ > θ_L · T_ave · (t − i) then return true  // Tentatively valid
8:          else if v⁻ − v⁺ > θ_L · T_ave · (t − i) then return false  // Tentatively invalid
9:          else // Layer margin too small
10:             return coin_t.  // Common coin at layer t
11:         end if
12:     end if
13: end function
14: function CountVotes(B: block, s: start layer, Ctx: context)
15:     Denote i the layer of block B.
16:     v⁺ ← 0
17:     v⁻ ← 0
18:     for all blocks B′ such that layer_B ≥ s and IsVotingBlock(B, Ctx) do
19:         if B′ voted for B then
20:             v⁺ ← v⁺ + w(B′)  // w(B′) denotes the voting weight of B′
21:         else if B′ voted against B then
22:             v⁻ ← v⁻ + w(B′)
23:         end if
24:     end for
25:     return (v⁺, v⁻)
26: end function
```

4.9.1 Non-Voting Blocks

One of the main differences between basing security on PoW and basing it on spacetime is that, unlike a PoW, the spacetime resource can't be bound to a specific "vote" (since we need to reuse the same storage space for multiple blocks, each potentially voting differently). This means that a malicious adversary can attempt to attack in the following way:

1. At time t, generate 2 layer-t blocks with the same node ID, one that votes x and the other $\neg x$. Note that this doesn't require extra resources. Publish only block x.

2. At some later time $t + s$, publish block $\neg x$.

Looking only at syntactic validity, both blocks are equally valid. Since we can't decide between the two syntactically, we consider *both* blocks to be "non-voting" in this case (they are still syntactically-valid for the purposes of the recursive syntactic validity rule).

No double-voting rule: If two blocks B, B' are both in layer i and have the same Node ID, then both are considered *non-voting*. Note that since one of the blocks can be published much later than the other,

this rule can be used to invalidate "historic" votes. However, our contextual validity rules will ensure that even if the adversary retroactively invalidates all of her votes, she cannot cause history to be reversed.

The pseudocode for determining whether a block is voting or nonvoting is function IsVotingBlock in fig. 4.7

4.10 Communication-Optimized Tortoise: Recursive Voting

The simple tortoise protocol described in section 4.9 requires every block to include a vote for *every* previous block. Explicitly doing so, however, would result in a *quadratic* increase in the size of the blockmesh as a function of time. Luckily, we don't have to include all votes explicitly. Instead, we can use the fact that the votes of a block B about a non-recent previous block X can be deterministically computed from the following information (where non-recent means $\text{layer}_B - \text{layer}_X > \text{hdist}$):

1. The value of $\text{coin}_{\text{layer}_B}$ (i.e., the value of the common coin at time B was generated according to the miner who generated it).

2. The votes of every syntactically-valid block in B's view about X.

Thus, we can explicitly encode only the votes for all recent blocks in B's view and its local view of $\text{coin}_{\text{layer}_B}$, and recursively compute item 2.

Security Intuition The intuition for security is that using the computed timestamp is permissible whenever the difference is less than δ, since honest nodes may differ by δ in their timestamps in any case.

4.11 Computation-Optimized Tortoise

While the method above does reduce the communication complexity, the computational complexity of the recursive computation is still high. We reduce the computational complexity using a combination of techniques.

4.11.1 Intuitive Computation-Optimized Protocol Description

The protocol has an optimistic "fast path" and a slower "secure fallback". The fast path will work as long as assumptions hold, but liveness is not guaranteed if assumptions are broken. When confirmation delay becomes too long, the protocol will automatically move to the secure fallback, which is more computationally intensive but will guarantee eventual consensus once assumptions are restored.

The fast path can be tried occasionally and the protocol can resume using the fast path once conditions are favorable.

The idea of the protocol is to use a dynamic programming inspired approach that looks roughly as follows. We would like to avoid having to either store all votes of previous blocks about other blocks or having to repetitively recompute them in order to compute newer blocks' implicit votes about older blocks.

We begin by observing that blocks vote explicitly about sets of blocks (called patterns) in older layers. There are three types of ways in which a block can vote about a pattern:

- It can point to a specific pattern in an older layer. These edges in the dependency graph are interpreted as positive votes. An honest block will point to the pattern in a layer that was the output of the hare protocol for that layer.

- If a block gives a positive vote about a pattern in a layer, then we interpret this as giving negative votes for all blocks in that layer it has not voted for in this way.

- It can explicitly abstain from giving a vote for a pattern in a layer. This happens whenever a block does not see the hare protocol as terminated for that layer.

Figure 4.8: Communication-Optimized Tortoise Validity

```
1: function IsTortoiseValid(B: block, t: current time, Ctx: context)
2:     // Identical to the "Simple" Tortoise (see fig. 4.7)
3: end function
4: function CountVotesByView(B: viewing block, X: target block)
5:     // Count the number of votes for X according to a block B
6:     Denote i ← layer_B
7:     v⁺ ← 0
8:     v⁻ ← 0
9:     for all blocks B′ such that layer_B′ ≥ i and IsVotingBlock(B′, Ctx) do
10:        vote ← ComputeVote(B′, B)  // B′s vote about B
11:        if vote = 1 then
12:            v⁺ ← v⁺ + 1
13:        else if vote = −1 then
14:            v⁻ ← v⁻ + 1
15:        end if
16:    end for
17:    return (v⁺, v⁻)
18: end function
19: function ComputeVote(B: voting block, X: target block)
20:    // Compute whether a block B voted for a block X
21:    // Returns a value in {+1, 0, −1} corresponding to for, neutral, against
22:    if layer_B ≤ layer_X then
23:        return 0
24:    else if layer_B − layer_X < hdist then
25:        if B explicitly votes for X then
26:            return 1
27:        else
28:            return −1
29:        end if
30:    else // Recursive Computation
31:        ...
32:    end if
33: end function
34: function IsVotingBlock(B: block, Ctx: context)
35:    // Identical to simple protocol, except using computed block timestamps
36: end function
```

Given that a layer is sufficiently far in the past, new honest blocks vote for the same pattern in that layer. This holds because all new blocks see that this layer's hare protocol has terminated and thus vote for the pattern obtained as output from the hare protocol. This means that, every layer will eventually have a unique 'good' pattern that garnered the majority of subsequent blocks' votes. Note that due to layers with dishonest majorities of blocks, it might take some subsequent layers until the good layers 'shine through' for an older layer.

Furthermore, good patterns point to all older blocks and thus one can compute their vote about them. Note however, that past pattern's votes about old blocks never change again and one can therefore compute them once and for all. Most importantly however, new honest blocks vote the same way about old blocks as (the most recent) good pattern (given that it had an honest majority).

Summarizing, good patterns in older layers eventually emerge and are the same as the hare protocol's output in that layer. Since all new blocks see (eventually) the same good patterns in past layers, an honest block B^* can calculate these newer blocks' implicit votes about old blocks using *the same good patterns* for all of them.

Our pseudocode and analysis formalizes this idea using a number of tables that it continuously updates upon seeing the blocks of a new layer and terminations of newer hare protocols. The code describes how these updates take place. Mainly, it shows how to keep track of newly emerging good patterns in past layers and how to calculate the opinions (votes about older blocks) of them using already computed entries in the tables.

4.11.2 Notation

Definition 4.3 (Vote-Pattern)**.** An *vote-pattern* is a layer index i and set of syntactically valid blocks in layer i. We denote $\mathsf{layer}(P)$ the layer index of pattern P.

Definition 4.4 (View)**.** The *view* $V(B)$ of a block B is defined as the set as the set of all blocks B' in layers $j < \mathsf{layer}(B)$ such that there exists a directed path of view edges from B to B'. The *view* $V(P)$ of a vote-pattern P is defined as $V(P) := \bigcup_{B \in P} V(B)$.

Definition 4.5 (Constrained View)**.** The *constrained view* $V(B)|_{>i}$ is defined as $V(B)|_{>i} := \{B \in V(B) \mid \mathsf{layer}(B) > i\}$. The *constrained view* $V(P)$ of a vote-pattern P is defined as $V(P)|_{>i} := \bigcup_{B \in P} V(B)|_{>i}$.

Definition 4.6 (Effective Voting Pattern)**.** A voting-pattern P is the *effective voting pattern* of a block B if B explicitly votes for P and does not explicitly vote for any pattern in layer $j > \mathsf{layer}(P)$.

To evaluate the implicit votes of block B, the computation optimized tortoise protocol only counts votes of the blocks in B's effective vote, i.e., for a pattern P in the most recent layer B voted about, and the blocks in P's view.

Definition 4.7 (Support)**.** A block B *supports* a pattern P if B explicitly or implicitly voted for all blocks in P, and no other blocks in layer $\mathsf{layer}(P)$.

Note that B only has *one* effective vote, but can support a (different) pattern in every layer. Furthermore, if B's effective vote is for P, B also supports P.

Definition 4.8 (Good Pattern)**.** A vote-pattern P is *B-good* if the weighted majority of blocks in $V(B)|_{>\mathsf{layer}(P)}$ support P and *B-bad* otherwise.

That is, of the blocks in our view that have i-vote-patterns, more than half the voting weight supported P. Note that a vote-pattern can be good in layer $i + 1$, and become bad in layer $i + 2$ and vice versa. However, we guarantee (w.h.p.) that eventually the honest vote pattern becomes good and stays good forever.

Definition 4.9 (Complete Pattern)**.** A vote-pattern P is *complete* if P is "confident" about the contextual validity of every block previous to P. Formally, P is complete if for every B with $\mathsf{layer}(\{B\}) < \mathsf{layer}(P)$ we have $\textsc{ConfidentOpinion}(P, B) \neq (0, 0)$ (see section 4.11.3).

Definition 4.10 (P_{base})**.** A vote-pattern P is called P_{base} if P is good and complete, and there is no other voting pattern in layer $j > \mathsf{layer}(P)$ is good and complete.

The new strategy is to count in the following way. We keep several tables, as described below:

- $T_{\texttt{pattern}}$: For a pattern P, the set of blocks $\mathcal{B}$ that makes up P, i.e., $T_{\texttt{pattern}}[P] = \mathcal{B}$.
- $T_{\texttt{support}}$: For a pattern P, the weight w of blocks that support this pattern, i.e., $T_{\texttt{support}}[P] = w$.
- $T_{\texttt{effectiveSupport}}$: For a pattern P, the weight w of blocks B s.t. $T_{\texttt{effective}}[B] = P$, i.e. $T_{\texttt{effectiveSupport}}[P] = w$
- $T_{\texttt{explicit}}$: For a block B, and a layer t, the pattern P that B explicitly votes for in layer t, i.e., $T_{\texttt{explicit}}[B, t] = P$.
- $T_{\texttt{effective}}$: For a block B, its effective voting pattern P, i.e., $T_{\texttt{effective}}[B] = P$.
- $T_{\texttt{tally}}$: For a pattern P and block B, the total weight of positive and negative votes (v^+, v^-) for B according to P's view, i.e., $T_{\texttt{tally}}[P, B] = (v^+, v^-)$. Note that $T_{\texttt{tally}}$ may not be defined for all patterns. When we don't have a tally for P, we denote $P \notin T_{\texttt{tally}}$ or $T_{\texttt{tally}}[P] = \perp$.
- $T_{\texttt{vote}}$: For a pattern P and block X, contains ConfidentOpinion(P, B) (see section 4.11.3)
- $T_{\texttt{pat-support}}$: For a pattern P and layer i, the pattern P' that P supports in layer i (or ? if it is still undefined). Pattern P supports P' in layer i if, according to P's "opinion", all the blocks in P' are contextually valid, and no other block is contextually valid in layer i.
- $T_{\texttt{good}}$: For a layer i the good pattern of layer i, i.e. $T_{\texttt{good}}[i] = P$ if and only if P is the good pattern for layer i.
- $T_{\texttt{correct}}$: For a pattern B and block X that B explicitly votes on, $T_{\texttt{correct}}[B, X] = (-1, 0)$ if B's implicit vote (on X) is $(1, 0)$, $T_{\texttt{correct}}[B, X] = (0, -1)$ if B's implicit vote is $(0, 1)$, and $T_{\texttt{correct}}[B, X] = (0, 0)$ otherwise. Note that this is a sparse table, with row B defined only if B's effective voting pattern is a good pattern.

4.11.3 Protocol Pseudocode

To compute B's opinion about a block X in layer i, we then use the following lookups:

```
1:  function ConfidentOpinion(P, X)
2:      Let (v⁺, v⁻) ← T_tally[P, X]
3:      if v⁺ > θ_G(Δ) then  // Δ := layer(P) − layer(X)
4:          return (1, 0)
5:      else if v⁻ > θ_G(Δ) then
6:          return (0, 1)
7:      else
8:          return (0, 0)
9:      end if
10: end function
```

This function return $(1, 0)$ if B is "confident" about X being contextually valid, $(0, 1)$ if B is confident about X being invalid, and $(0, 0)$ if B is not confident about X's validity.

We use the following procedure to update the tally of a pattern according to the good layers in its view.

```
1: procedure UpdatePatternTally(P_base, G, P)
```

2: // Update P's tally given a good pattern G with $\mathsf{layer}(G) < \mathsf{layer}(P)$. The idea is that P's tally was already initialized to the tally of P_{base}, it remains to add the votes of blocks that are in P's view, but not in P_{base}'s view.

$corr \leftarrow \vec{0}$ // Local correction vector
$effcount \leftarrow 0$ // Weight of blocks whose effective vote is for G
for all $B \in P \cup V(P)|_{>\mathsf{layer}(P_{\text{base}})}$ s.t. B's effective voting pattern is G **do**
$corr \leftarrow corr + weight(B) \cdot T_{\texttt{correct}}[B]$
$effcount \leftarrow effcount + weight(B)$;
end for
$T_{\texttt{tally}}[P] \leftarrow T_{\texttt{tally}}[P] + T_{\texttt{vote}}[G] \cdot effcount + corr$
end procedure

Since blocks may vote both explicitly and implicitly, we need to make sure that we do not count these votes twice. The following procedure is responsible for that.

procedure UpdateCorrectionVectors(P) // Update the correction vectors for all blocks whose effective vote is P
for all B whose effective vote is P **do**
$T_{\texttt{correct}}[B] \leftarrow \vec{0}$
for all blocks $X \in V(P)$ **do**
if $T_{\texttt{explicit}}[B, X] \neq 0$ **then**
$T_{\texttt{correct}}[B, X] \leftarrow -T_{\texttt{vote}}[P, X]$
end if
end for
end for
end procedure

To update the tables, we use the following recursive strategy for every new layer i.

procedure UpdateTables($\mathcal{B}, i$) // Let $\mathcal{B}$ be the new blocks of layer i.
do
P_{base} is the latest pattern which is both complete and good.
$\ell \leftarrow i$ // k is layer of the earliest "newly good" pattern
for all $j \in [\max\{\mathsf{layer}(P_{\text{base}}) + 1, i - w\}, i)$ **do** // w is window size
// This loop computes the updated support for all patterns in the window
$S_{\text{updated}} \leftarrow \emptyset$ // S_{updated} contains the set of patterns whose support has been updated, so they could become newly good
for all $B \in \mathcal{B}$ **do**
if B *explicitly* votes about pattern P in layer j **then**
$T_{\texttt{explicit}}[B, j] \leftarrow P$
$T_{\texttt{support}}[P] \leftarrow T_{\texttt{support}}[P] + weight(B)$
$S_{\text{updated}} \leftarrow S_{\text{updated}} \cup P$
else
Let $P \leftarrow T_{\texttt{pat-support}}[T_{\texttt{effective}}[B], j]$ // Pattern P that B *implicitly* supports in layer j
if $P \neq ?$ **then**
$T_{\texttt{support}}[P] \leftarrow T_{\texttt{support}}[P] + weight(B)$
$S_{\text{updated}} \leftarrow S_{\text{updated}} \cup P$
end if
end if
end for
for all patterns $P \in S_{\text{updated}}$ such that $T_{\texttt{good}}[\mathsf{layer}(P)] \neq P$ **do**
// Go over patterns whose support has been updated, check if any have become good
if $T_{\texttt{support}}[P] > \frac{1}{2}\left|V(B^*)|_{>\mathsf{layer}(P)}\right|$ **then** // a majority of blocks support P
$T_{\texttt{good}}[\mathsf{layer}(P)] \leftarrow P$
if $\mathsf{layer}(P) < \ell$ **then**
$\ell \leftarrow \mathsf{layer}(P)$ // Keep track of the minimal newly good layer
end if

48: **end if**
49: **end for**
50: **end for**
51: **for all** $j \in [\ell, i)$ **do**
52: $P \leftarrow T_{\texttt{good}}[j]$
53: $T_{\texttt{tally}}[P] \leftarrow T_{\texttt{tally}}[P_{\text{base}}]$
54: **for** $k \in [\mathsf{layer}(P_{\text{base}}), \mathsf{layer}(P))$ **do**
55: $G_k \leftarrow T_{\texttt{good}}[k]$
56: **if** $G_k \neq \perp$ **then**
57: UpdatePatternTally$(P_{\text{base}}, G_k, P)$
58: **end if**
59: **end for**
60: **for all** $B \in P \cup V(P)|_{>\mathsf{layer}(P_{\text{base}})}$ **do**
61: Let $V^\dagger$ be a vector, indexed by block, such that for block X in layer i:
62: **if** $T_{\texttt{explicit}}[B, i] \neq \perp$ **then**
63: If $X \in T_{\texttt{explicit}}[B, i]$ then $V^\dagger_X = +1$ contains $+1$, otherwise $V^\dagger_X = -1$
64: **else** // $T_{\texttt{explicit}}[B, i] = 0$
65: $V^\dagger_X = 0$
66: **end if**
67: $T_{\texttt{tally}}[P] \leftarrow T_{\texttt{tally}}[P] + \mathit{weight}(B) \cdot V^\dagger$
68: **end for**
69: $T_{\texttt{vote}}\left[P, \vec{X}\right] \leftarrow$ ConfidentOpinion$(P, \vec{X})$ // $\vec{X}$ is the vector of all blocks previous to P.
70: Compute $T_{\texttt{pat-support}}\left[P, \vec{j}\right]$ // $\vec{j}$ is the vector of all layers previous to $\mathsf{layer}(P)$
71: UpdateCorrectionVectors(P)
72: **if** P is complete **then**
73: $P_{\text{base}} \leftarrow P$
74: **end if**
75: **end for**
76: **while** A newly good pattern has been found in the current iteration
77: **end procedure**

4.11.4 Analysis

Theorem 4.11 (Soundness, informal)**.** *If an honest block A votes v about an honest block B in the optimized tortoise protocol, then it votes the same way in the unoptimized one.*

Proof Sketch. Suppose that A sees a good pattern P in some layer i and votes about B implicitly according to P. Since good patterns include all honest parties blocks and there is an honest majority in every layer, the implicit votes for block B corresponding to P amounts to the same as tallying the votes of layer i as done in the unoptimized protocol. By a recursive argument, good patterns are the same in the optimized and unoptimized versions of the protocol. Hence, honest blocks vote the same in both versions about other honest blocks. □

Theorem 4.12 (Completeness, informal)**.** *As long as the Hare protocol properly works, every honest party P eventually has an opinion about v about every block B in the optimized tortoise protocol. Moreover, if layers are sufficiently large, P has an opinion about any block after at most one layer.*

Proof Sketch. Let B be an arbitrary block and denote $i = \text{layer}_B$. According to the optimized voting rule, we only need to argue that eventually P will consider some pattern good for layer i (once P identifies a good pattern for layer i, it has opinions about every block in layer i). To see this will be the case, observe that as long as the Hare protocol works properly, honest parties vote the same way about patterns in every layer. Since honest parties have a majority, this ensures that the pattern they vote for in any given layer, will eventually become the good pattern for that layer. Moreover, if layers are sufficiently large,

a round of explicit votes from the Hare protocol (which corresponds to one layer) suffices to cross the threshold to make the pattern good. □

5 Tortoise Protocol Security Analysis

5.1 Overview

Our security analysis is in several steps:

1. (cf. section 5.2) We prove that the adversary cannot "concentrate" its weight in any small set of layers (i.e., we show that the measures described in section 4.5.2 to prevent concentration attacks are sufficient).

2. (cf. section 5.3) We describe and analyze a simplified model that we call "graded consensus with disruption tokens". Loosely, in this model we ignore most of the underlying mechanics of the protocol, and concentrate on the validity of blocks. Every honest party maintains a tuple for every block it has seen: its "opinion" about the block (i.e., whether or not the block is valid) and its "grade" (i.e., how confident it is about its opinion).

   We show that if a protocol in this model satisfies a set of simple properties, then it will achieve some strong security properties. In particular, honest parties will reach consensus about the validity of all blocks and history will be irreversible. If all honestly-generated blocks are considered valid by all honest parties at the time they are generated, then these security properties will also imply that all honestly-generated blocks will be considered valid in the final consensus.

3. (cf. section 5.4) We show a bijective mapping from the real-world protocol to the simplified model, and prove that this mapping satisfies the required properties. This implies that the real-world protocol also has the desired security properties. Note that since we do not use the mapping to construct an adversary, this mapping is not required to be *efficient*; the mere existence of such a mapping is sufficient to guarantee security.

4. (cf. section 5.5) We prove that every honest block is always considered valid in the final consensus.

5. (cf. section 5.6) We show how the security properties that are achieved imply the standard notions of blockchain security, as defined by Pass et al. [27].

5.1.1 Opinions on History

The core of any distributed ledger is consensus on history. Our goal is to show that there is agreement about an ordered list of blocks. However, it is sufficient to consider separately the consensus on every block in history (as explained in section 1.2.1).

In our analysis, we define a party's opinion about a block B to be 1 if the party considers B valid (i.e., included in the ledger) and -1 if not. We conflate the opinions of *parties* with the opinions of *blocks*: the opinion of a block is the opinion of the party that generated that block at the time it was generated.

Every party has an opinion about every block in its view (and executes the consensus algorithm in parallel for all of them). In this section, except where we explicitly state otherwise, we are considering the opinions of parties/blocks only about a specific block B.

For simplicity, we will describe the protocol as if each block explicitly contains the complete list of opinions held by the party that generated the block; in the actual protocol these are computed from the view of the block.

5.2 Bounding the adversarial weight concentration

Figure 5.1: k-PoET Security Game

1: $X \leftarrow P^H()$ // $X \in \{0,1\}^\kappa$
2: **if** $H[X] \neq \perp$ **then**
3: **return** 0
4: **end if**
5: $\pi \leftarrow P^H(X)$ // Let Q be the longest sequence of sequential queries to H over the course of the game
6: **return** $verify(\pi, X) \wedge Q < k$

Figure 5.2: Composed PoET Construction CompPoET

1: **function** PROVE(ch:challenge)
2: $D_1 \leftarrow k_1$
3: $D_2 \leftarrow k_2$
4: $\pi_1 \leftarrow \text{PoET}(ch, D_1)$
5: $\pi_2 \leftarrow \text{PoET}(H(\pi_1), D_2)$
6: Return (π_1, π_2)
7: **end function**
8: **function** VERIFY($\pi = (\pi_1, \pi_2)$: proof, ch : challenge)
9: $D_1 \leftarrow k_1$
10: $D_2 \leftarrow k_2$
11: **if** $\text{PoET}(\pi_1, ch, D_1)$ **then**
12: **return** $\text{PoET}(\pi_2, H(\pi_1), D_2)$
13: **else**
14: **return** 0
15: **end if**
16: **end function**

Lemma 5.1. *Consider the construction of* CompPoET *in section 5.2. If* PoET *is a secure proof of elapsed time for durations k_1 and k_2 and* PoET *(in the sense of section 5.2), then* CompPoET *is a secure proof of elapsed time for duration $k_1 + k_2$.*

Proof. (Sketch). Consider an adversary $\mathcal{A}$ that breaks the security of the CompPoET, i.e., wins the game in section 5.2 with respect to CompPoET. This means that it succeeds in computing the proof π with fewer than $k = k_1 + k_2$ sequential hash queries to the random oracle H. $\mathcal{A}$ first outputs a challenge ch. Since $\mathcal{A}$ is successful, we can assume that at the beginning of the security game, $H[ch] = \perp$ and thus any value computed prior to this point from H is independent of π. Let us first consider the case that $\mathcal{A}$ asks all queries to H that are related to π_1 before making any queries that are related to π_2. In this case, it must make fewer than k_1 sequential queries to H for the computation of π_1 or fewer than k_2 sequential queries to H for the computation of π_2. Either scenario yields a contradiction to the assumption that PoET is a secure PoET (for durations k_1, k_2). Now consider the case that $\mathcal{A}$ starts to make queries to H related to π_2 before k_1 queries to H have been made since $\mathcal{A}$ has output ch. Since by assumption $\mathcal{A}$ can not compute π_1 with fewer than k_1 queries to H, any output from H must be uniformly distributed and independent from the value of $H(\pi)$ and thus provide no information about π_2 before $\mathcal{A}$ has made k_1 queries to H. □

Lemma 5.2. *Let the honest parties' tick be t. Then there can't exist an ATX with start tick $t' > t$.*

Proof. To prove this claim, it suffices to notice that by the previous lemma, any ATX holds a PoET (measuring elapsed time starting from the genesis block). Thus if there exists an ATX with start tick $t' > t$, this ATX must hold a PoET that violates the PoET security game. □

Corollary 5.3. *Let B be an honestly generated block in epoch z. Then for every block X in epoch $z' \leq z$, the total weight of adversarial votes for X in B's view is at most $(z - z' + 1)qn$.*

Proof. By lemma 5.2, before the start of epoch z', the adversary cannot generate an ATX with "valid" work tick for epoch z' (i.e., a work tick that is consistent with the honest tick in epoch z'). Every block referencing an ATX with a lower work tick will have weight 0 in B's view (due to the neutral vote rule). Since, by our assumption on the adversary's spacetime, the total spacetime per epoch controlled by the adversary is at most qn, the adversary's voting weight in the epochs $z', \ldots, z$ is bounded by $(z - z' + 1)qn$. □

Lemma 5.4. *Let $\ell > 1$ be the bad beacon postponement distance. Let z be an epoch, $0 < k$, and let H_k the total adversarial voting weight over k consecutive layers in epochs $z, \ldots, z + \ell$, according to an honest block in epoch $z + \ell$. Then for all $\delta > 0$, $\Pr[H_k > (1 + \delta)\mu] \leq 2(1 + 2^{-\kappa})e^{-\frac{qn(\delta qn)^2}{k^2}}$, where $\mu = k \cdot q \cdot n/\Delta_{epoch} = k \cdot q \cdot T_{ave}$.*

Proof. First, note that all blocks with non-zero weight must use the honest epoch beacons since they are all within the bad beacon postponement distance (section 4.5.2). Second, in every full epoch, the total adversarial weight is exactly qn; this follows from corollary 5.3. thus, we need only consider the first and last epochs that may be partial. We assume that the beacons for epoch z and $z + \ell$ behave as i.i.d uniform variables. For every ID id and its i-th block, we can associate independent random variables $B_{id,i} \in [0, 1]$ which behaves as follows: $B_{id,i} = 0$ if id's i-th block is *not* eligible to vote in the interval from layers j through $j + k$ of epochs z or $z + \ell$ and $B_{id,i} > 0$ represents the associated block's voting weight if that block is eligible to vote in this interval. Thus, we can write $H_k = \sum_{id,i} B_{id,i}$. Let $p = k/\Delta_{\text{epoch}}$ denote the probability for a given block to be eligible to vote during layers j through $j + k$. Then $\mu = pq \cdot n$. Assuming that each identity has at least one block, the Hoeffding bound asserts that $\Pr[H_k > (1 + \delta)\mu] \leq e^{-\frac{qn(\delta qn)^2}{k^2}}$. If we assume that instead the beacon has low influence, then we obtain instead the slightly weaker bound $\Pr[H_k > (1 + \delta)\mu] \leq (1 + 2^{-\kappa})e^{-\frac{qn(\delta qn)^2}{k^2}}$. Finally, we can lift the restriction of all k layers being in the same epoch by bounding the weight of each interval separately and then applying the union bound. If there are k_1 layers in epoch z and k_2 layers in epoch $z + \ell$, this yields

$\Pr[H_k > (1+\delta)\mu] \leq \Pr[H_{k_1} > (1+\delta)\mu] + \Pr[H_{k_2} > (1+\delta)\mu] \leq (1+2^{-\kappa})e^{-\frac{qn(\delta qn)^2}{k_1^2}} + (1+2^{-\kappa})e^{-\frac{qn(\delta qn)^2}{k_2^2}} \leq 2(1+2^{-\kappa})e^{-\frac{qn(\delta qn)^2}{k^2}}$ the probability term in the lemma statement. □

Lemma 5.5. *Let $\ell > 1$ be the bad beacon postponement distance. For every time t, $k > 0$, every block X and every constant $\delta > 0$ the probability that the total voting weight (about X) of adversarial blocks received by honest parties in the interval $t, ..., t+k$ is more than $\left(\frac{2+\delta}{\ell-1} + 1\right) kqT_{ave}$ is less than $f(k,n)$, where f is a negligible function in both k and $n = T_{ave} \cdot \Delta_{epoch}$.*

Proof. Assume w.l.o.g. that $\text{layer}_X < t$ (otherwise the adversary cannot vote at all for X in the interval $t, \ldots, \text{layer}_X$), and let t be in epoch z. Denote $r = k \mod \Delta_{\text{epoch}}$, such that $k = \ell\Delta_{\text{epoch}} + r$. We distinguish two cases:

Case 1: $t + k$ is in epoch $z' \leq z + \ell$. In this case the claim follows immediately from lemma 5.4.

Case 2: $t + k$ is in epoch $z' > z + \ell$ (hence $k \geq (\ell - 1) \cdot \Delta_{\text{epoch}}$). Let $t + s$ be the last layer of epoch z. Layers $t, \ldots, t + s$ contribute at most qn voting weight (this is tight, since the adversary can concentrate all its weight for epoch z in layer $t + s$ by choosing its beacon maliciously, separately for each identity). Each of the $\left\lfloor (k-s)/\Delta_{\text{epoch}} \right\rfloor$ full epochs between t and $t + k$ also contribute at most qn weight each. Let $r = (k - s) \mod \Delta_{\text{epoch}}$. By lemma 5.4, the last r layers contribute at most $(1+\delta)rqT_{\text{ave}}$ weight except with negligible probability.

Thus, in total the weight contributed by these k layers is at most

$$\begin{aligned} qn + \left\lfloor (k-s)/\Delta_{\text{epoch}} \right\rfloor qn + (1+\delta)rqT_{\text{ave}} &\leq \left((1 + \left\lfloor k/\Delta_{\text{epoch}} \right\rfloor)\Delta_{\text{epoch}} + (1+\delta)\Delta_{\text{epoch}}\right) qT_{\text{ave}} \\ &= \left((1 + \left\lfloor k/\Delta_{\text{epoch}} \right\rfloor)\Delta_{\text{epoch}}/k + (1+\delta)\Delta_{\text{epoch}}/k\right) kqT_{\text{ave}} \\ &= \left((2+\delta)\Delta_{\text{epoch}}/k + \frac{\left\lfloor k/\Delta_{\text{epoch}} \right\rfloor}{k/\Delta_{\text{epoch}}})\right) kqT_{\text{ave}} \\ &\leq \left(\frac{2+\delta}{\ell-1} + 1\right) kqT_{\text{ave}} \end{aligned}$$

□

5.3 Graded Consensus with Disruption Tokens

We will use a notion of *graded consensus* between honest parties, and leverage that to achieve full consensus. We extend the standard notion of graded consensus with *disruption tokens*: each token can be spent once by the adversary to violate the basic graded consensus guarantees (although we call them *tokens*, the adversary can spend part of a token—one can think of the tokens as the basic unit of a continuous disruption budget). Disruption tokens come in two flavors, positive and negative; these cancel, so spending a positive and a negative token at the same time has no effect.

Disruption tokens are also specific to the block B (i.e., spending a token for block B is independent of spending a token for block $B' \neq B$).[8] Opinions about different blocks can be independent (but for every block, the graded opinions satisfy the properties described below).

In our graded consensus, each party has an opinion $x \in \{\pm 1\}$ and a non-negative grade $g \in \{0\} \cup \mathbb{N}$. We say that party's *graded opinion* is $x \cdot g \in \mathbb{Z}$. We denote T_t^+ (resp. T_t^-) the number of positive (resp. negative) disruption tokens spent by the adversary at time t (we write T_t^x to mean T_t^+ when $x = 1$ and T_t^- when $x = -1$).

We will require the following properties to hold in order to guarantee consensus:

1. If an honest block at time t has graded opinion $x \cdot g$, then for every honest block at time $t + 1$ with graded opinion $x' \cdot g'$ one of the following holds:

[8] In reality, this gives extra power to the adversary, since tokens cannot be spent entirely independently.

- $x = 1$ and $x'g' - xg \geq -(T_t^{-x} + T_{t+1}^{-x} + 1)$.
- $x = -1$ and $xg - x'g' \geq -(T_t^{-x} + T_{t+1}^{-x} + 1)$.

In other words,

$$x(x'g' - xg) \geq -(T_t^{-x} + T_{t+1}^{-x} + 1).$$

2. The total number of disruption tokens of each flavor spent by the adversary up to time t is less than $q \cdot (t - \text{layer}_B) + T_0$; in this bound $T_0 > 0$ models cases where the adversary gained some unfair initial advantage (i.e., if our assumptions hold, $T_0 = 0$ and the adversary can spend less than one positive and one negative disruption token for every $1/q$ layers since the publication of B).

3. Suppose all honest parties at time t have the same opinion x and grade at least $g_t \geq 1$. Then for every honest party at time $t + 1$ with graded opinion $x' \cdot g'$, it holds that $x \cdot x' \cdot g' \geq g_t + 1 - T_{t+1}^{-x}$. That is, every honest party will have the same opinion and a least grade will be higher, unless the adversary spends disruption tokens of the flavor opposite that opinion.

For self-healing, we will require three additional properties. For every block B and layer $t > \text{layer}_B$, we consider the layer either *B-special* or *B-regular*. Specialness is a randomized property that must satisfy:

4. For all B, t, the probability that layer t is B-special is at least p.

5. For all B and every set of layer indices T, the events $\{t \text{ is } B\text{-special}\}_{t \in T}$ are independent.

6. For all B, t, if t is B-special then either the adversary spent a B-token at time t or all honest blocks have the same opinion of B at time $t + 1$ and grade at least 1.

Note that where the block B is clear, we omit it and say simply that layer t is special.

Figure 5.3 describes the tortoise validity rules in this graded consensus model.

Figure 5.3: Tortoise Validity (Abstract)

```
1: function IsTortoiseValid(B: block, t: current time)
2:     if opinion of B is x with grade at greater than 2 + (t − layer_B) · q then
3:         return x (confidently)
4:     else if opinion of B is x with grade at least 1 then
5:         return x (tentatively)
6:     else
7:         return local coin (tentatively)
8:     end if
9: end function
```

Definition 5.6 (Confident Consensus)**.** We say that B is in *confident consensus* x at time t if, for every honest party, IsTortoiseValid(B, t) returns x confidently.

For all $t > \text{layer}_B$, denote $\Delta_t \doteq (t - \text{layer}_B)$. Let $\theta_G \doteq 2 + q \cdot \Delta_t + T_0$. We'll use the following simple claim:

Claim 5.7. *For all $i \geq 0$, $\theta_G + i - 1 - \max_{\times \in \{+,-\}} \sum_{j=0}^{i} T_{t+j}^{\times} > 1$.*

Proof. By property 2, for all $i \geq 0$,

$$\max_{\times \in \{+,-\}} \sum_{j=0}^{i} T_{t+j}^{\times} < T_0 + q \cdot (t + i - \text{layer}_B) = T_0 + q\Delta_t + q \cdot i\,.$$

Thus,

$$\begin{aligned}\theta_G + i - 1 - \max_{\times\in\{+,-\}} \sum_{j=0}^{i} T_{t+j}^{\times} &> \theta_G + i - 1 - T_0 - q\Delta_t - q \cdot i \\ &= 2 + q \cdot \Delta_t + T_0 + i - 1 - T_0 - q\Delta_t - q \cdot i \\ &= 1 + i - q \cdot i = 1 + (1-q)i \geq 1\end{aligned}$$

□

Denote g_i the minimal grade for any honest party at time i (regardless of opinion).

Lemma 5.8. *For all $t > \text{layer}_B$, if any honest party at time t has opinion x and grade $g \geq \theta_G$, then for all $i \geq 1$, every honest party at time $t + i$ will have opinion x and*

$$g_{t+i} \geq g + i - 1 - \sum_{j=0}^{i} T_{t+j}^{-x} \geq 1 \,.$$

Proof. The proof is by induction on i (the second inequality follows directly from claim 5.7).

1. Base case ($i = 1$): Let $x'g_{t+1}$ be the graded opinion of some honest party at time $t+1$. By property 1 $x(x'g_{t+1} - xg) \geq -(T_t^{-x} + T_{t+1}^{-x} + 1)$, hence if $x = 1$ we get that $x'g_{t+1} \geq g - T_t^{-x} - T_{t+1}^{-x} - 1 \geq \theta_G - T_t^{-x} - T_{t+1}^{-x} - 1 > 0$, where the last inequality follows from claim 5.7. On the other hand, if $x = -1$, we get $-x'g_{t+1} \geq g - T_t^{-x} - T_{t+1}^{-x} - 1 > 0$. In both cases, since the left-hand side of the inequality is an integer (and g_{t+1} is non-negative), it follows that $x' = x$ as well as $g_{t+1} \geq 1$.

2. Assume the induction hypothesis holds for i.

3. To show it holds for $i + 1$: By the induction hypothesis all honest parties at time $t + i$ have opinion x, and $g_{t+i} \geq g + i - 1 - \sum_{j=0}^{i} T_{t+j}^{-x} \geq 1$. Thus, using property 3,

$$\begin{aligned}xx'g_{t+i+1} &\geq g_{t+i} + 1 - T_{t+i+1}^{-x} \\ &\geq g + i - 1 - \sum_{j=0}^{i} T_{t+j}^{-x} + 1 - T_{t+i+1}^{-x} \\ &\geq \theta_G + (i+1) - 1 - \sum_{j=0}^{i+1} T_{t+j}^{-x} \\ &\geq 1\end{aligned}$$

(where the final inequality is due to claim 5.7). Thus $x' = x$ and the claim follows.

□

Corollary 5.9. *For all $t > \text{layer}_B$, if any honest party at time t has opinion x and grade $g \geq \theta_G$, then for every $z \geq \frac{1}{1-2q} \cdot (1 + q\Delta_t)$, B will be in confident consensus on x at time $t + z$*

Proof. By lemma 5.8, at time $t + z$ every honest party will have opinion x and grade at least

$$\begin{aligned}g_{t+z} &\geq g + z - 1 - \sum_{j=0}^{z} T_{t+j}^{-x} \\ &\geq 2 + q(t - \text{layer}_B) + T_0 + z - 1 - \sum_{j=0}^{z} T_{t+j}^{-x}\end{aligned}$$

By property 2, $\sum_{j=0}^{z} T_{t+j}^{-x} < q(t+z-\text{layer}_B) + T_0$

$$\begin{aligned}
&> 2 + q(t-\text{layer}_B) + T_0 + z - 1 - q(t+z-\text{layer}_B) - T_0 \\
&= 2 + q(t+z-\text{layer}_B) - q\cdot z + z - 1 - q(t+z-\text{layer}_B) \\
&= 2 + q(t+z-\text{layer}_B) + (1-2q)z - 1 - q(t-\text{layer}_B)
\end{aligned}$$

Since $z \geq \frac{1}{1-2q}\cdot(1+q\Delta_t)$, and hence $(1-2q)\cdot z - 1 - q\Delta_t \geq 0$:

$$\geq 2 + q\cdot(t+z-\text{layer}_B)\,.$$

Thus, IsTortoiseValid$(B, t+z)$ will return x confidently for all honest parties. □

5.3.1 Proving Self-Healing

Our proof of eventual consensus will, at a high level, consist of two parts. In claim 5.11, we will show that causing honest parties to disagree in layer $t+s$ requires the adversary to spend many tokens (where many depends on the number of special indices), hence after some point (at which the adversary will run out of tokens), all honest parties will agree. We then use lemma 5.12 (whose proof is very similar to lemma 5.8) to show that once this occurs, the honest parties' confidence grows monotonically until confident consensus is reached.

The following claim lower bounds the amount the adversary must "pay" to cause disagreement after consecutive layers in which honest parties agree.

Claim 5.10. *For all $t \geq 0$, if for all $i \in \{0,\ldots,k-1\}$ it holds that at time $t+i$ all honest parties agree on B with grade at least 1 (but might have different opinions at different times), and at time $t+k$ all honest parties* do not *agree on B, then $\sum_{i=1}^{k}(T_{t+i}^{+} + T_{t+i}^{-}) \geq k$.*

Proof. We'll first prove the following statement by induction on i: For all $i \in \{1,\ldots,k-1\}$, at time $t+i$ all honest parties have grade at least $1 + i - \sum_{j=1}^{i}(T_{t+j}^{+} + T_{t+j}^{-})$.

For $i = 1$, this follows immediately from property 3: at time t, all honest parties agree and have grade at least 1, therefore at time $t+1$ they must agree and have grade at least $2 - \max\{T_t^{+}, T_t^{-}\} \geq 2 - (T_t^{+} + T_t^{-})$. For $i+1 < k$: by the induction hypothesis, at time $t+i$ all honest parties have grade at least $g_{t+i} \geq 1+i-\sum_{j=1}^{i}(T_{t+j}^{+}+T_{t+j}^{-})$ (and by the claim condition $g_{t+i} \geq 1$). Then by property 3, at time $t+i+1$, all honest parties have grade at least $g_{t+i}+1-\max\left\{T_{t+i+1}^{+}, T_{t+i+1}^{-}\right\} \geq g_i+1-(T_{t+i+1}^{+}+T_{t+i+1}^{-}) \geq i+1-\sum_{j=1}^{i+1}(T_{t+j}^{+}+T_{t+j}^{-})$.

Thus, at time $t+k-1$, it holds that all honest parties agree on some x (assume w.l.o.g that $x = 1$) and have grade at least $g_{t+k-1} = k - 1 - \sum_{i=1}^{k-1}(T_{t+i}^{+} + T_{t+i}^{-})$. Let g_{t+k} be the minimal honest grade at time $t+k$. Since there exist two honest parties that disagree, some honest party must have opinion $-x = -1$, thus its graded opinion can be at most $-g_{t+k} \leq 0$, while all honest parties at time $t+k-1$ have graded opinion at least g_{t+k-1}. Thus, by property 3,

$$0 \geq -g_{t+k} \geq g_{t+k-1} + 1 - (T_{t+k}^{+} + T_{t+k}^{-}) \geq k - \sum_{i=1}^{k}(T_{t+i}^{+} + T_{t+i}^{-}) \Leftrightarrow \sum_{i=1}^{k}(T_{t+i}^{+} + T_{t+i}^{-}) \geq k\,.$$

□

Using claim 5.11, we show that to cause disagreement in any round, the adversary must pay an amount matching the number of special layers preceding that round.

Claim 5.11. *Let s be an index such that at time $t+s$ honest parties did not all have the same opinion of B. Denote $S_s = \{i \leq s | t+i \text{ is special}\}$. Then $\sum_{i=0}^{s}(T_{t+i}^{+} + T_{t+i}^{-}) \geq |S_s|$.*

Proof. The idea behind the proof is that every consecutive sequence of j "agreeing" indices (i.e., for which all honest parties agree) requires the adversary to spend at least j tokens in order to make the next index a "disagreeing" index. Also, every *special* disagreeing index also requires a token, otherwise (by property 6) it would be agreeing. So basically the adversary pays for every index except disagreeing non-special ones, and in particular pays for every special index, whether it is an agreeing index or not.

Formally, denote $Z = z_1, \ldots, z_n$ the set of indices such that $z \in Z$ iff $z \leq s$ and at time $t + z$ honest parties did not all have the same opinion of B (i.e., Z is the set of disagreeing indices). Let $R_1, \ldots, R_n$ be disjoint consecutive sets of *agreeing* indices; that is, $i \in R_j$ iff $i \notin Z$ and either $i + 1 \in R_j$ or $i + 1 = z_j$ (R_j is empty if $z_{j-1} + 1 = z_j$). Let $R_j^+ = R_j \cup z_j$ if $|R_j| > 0$ and $R_j^+ = \emptyset$ if $|R_j| = 0$.

By claim 5.10, for all j it holds that $\sum_{i \in R_j^+}(T_{t+i}^+ + T_{t+i}^-) \geq |R_j^+|$. For $i \in S_s \setminus \dot{\bigcup}_{j=1}^n R_j^+$, time $t + i$ is special (since $i \in S_s$) and honest parties do not agree on B at time $t + i$ (otherwise i would be in some R_j). Thus, by property 6, $(T_{t+i}^+ + T_{t+i}^-) \geq 1$. Thus,

$$\sum_{i \leq s}(T_{t+i}^+ + T_{t+i}^-) \geq \sum_{i \in \dot{\bigcup}_{j=1}^n R_j^+}(T_{t+i}^+ + T_{t+i}^-) + \sum_{i \in S_s \setminus \dot{\bigcup}_{j=1}^n R_j^+}(T_{t+i}^+ + T_{t+i}^-) \geq \sum_{j=1}^n |R_j^+| + |S_s \setminus \dot{\bigcup_{j=1}^n} R_j^+| \geq |S_s|$$

□

Lemma 5.12. *For all $t > layer_B$, and all $i \geq 0$, if, in the entire interval $t, \ldots, t + i$, every honest party has opinion x, then every honest party at time $t + i + 1$ will have grade $g_{t+i+1} \geq g_t + i + 1 - \sum_{j=0}^i T_{t+j+1}^{-x}$.*

Proof. The proof is by induction on i.

1. Holds trivally for the base case ($i = 0$).
2. Assume the induction hypothesis holds for i.
3. To show it holds for $i + 1$: From the induction hypothesis all honest parties have the same opinion at time $t + i + 1$, hence by property 3,

   $$g_{t+i+2} \geq g_{t+i+1} + 1 - T_{t+i+2}^{-x}$$

   And using the induction hypothesis:

   $$\geq g_t + i + 1 - \sum_{j=0}^{i} T_{t+j+1}^{-x} + 1 - T_{t+i+2}^{-x}$$
   $$= g_t + (i + 2) - \sum_{j=0}^{i+1} T_{t+j+1}^{-x}$$

□

Corollary 5.13. *Let $t > layer_B$ and $i \geq 0$, and suppose that in the entire interval $t, \ldots, t + i$ every honest party has opinion x. Then, if $g_t + i + 1 > \sum_{j=0}^i T_{t+j+1}^{-x}$, every honest party has opinion x at time $t + i + 1$.*

Proof. Let x' be the opinion of some honest party at time $t + i + 1$. From property 3 and the proof above, we have

$$xx' g_{t+i+1} \geq g_{t+i} + 1 - T_{t+i+1}^{-x} \geq g_t + i + 1 - \sum_{j=0}^{i} T_{t+j+1}^{-x} > 0\,.$$

□

Lemma 5.14 (Convergence with Disruption (Informal))**.** *For initial supply of disruption tokens T^* there exists $z > 0$, such that for every block B at time t, and any initial distribution of graded opinions at time t, such that the probability that B is not in confident consensus at time $t + z$ is negligible in the security parameter.*

Proof Sketch. Let D_s be the event that at time $t + s$ not all honest parties agree on B, and denote $\rho_s = \Pr[D_s]$ (where the probability is over the choice of special indices and the random coins of the adversary).

By claim 5.11, the event D_s implies that $|S_s| \leq \sum_{i=0}^{s}(T_{t+i}^{+} + T_{t+i}^{-})$, and by property 2,

$$\sum_{i=0}^{s}(T_{t+i}^{+} + T_{t+i}^{-}) \leq T_0 + 2q \cdot (t + s - \text{layer}_B) .$$

hence D_s implies

$$|S_s| \leq \sum_{i=0}^{s}(T_{t+i}^{+} + T_{t+i}^{-}) \leq T_0 + 2q \cdot (t + s - \text{layer}_B) .$$

Thus $\rho_s \leq \Pr[|S_s| \leq T_0 + 2q \cdot (t + s - \text{layer}_B)]$. Since every layer is independently special with probability p, $E[|S_s|] = p \cdot s$, and by Chernoff we have for all $\delta \in (0, 1)$

$$\Pr[|S_s| < (1 - \delta)p \cdot s] < e^{-\frac{\delta^2 \cdot p}{2} \cdot s}$$

Thus, when $(1 - \delta)p > 2q$ and $s > \frac{T_0 + 2q\Delta_t}{(1-\delta)p - 2q}$ (implying $T_0 + 2q \cdot (t + s - \text{layer}_B) \leq (1 - \delta)p \cdot s$), it follows that $\rho_s < e^{-\frac{\delta^2 \cdot p}{2} \cdot s}$.

To show that w.h.p. there is a maximal index s^* for which honest parties disagree, we can bound for arbitrary $s^* > \frac{T_0 + 2q\Delta_t}{(1-\delta)p - 2q}$ the probability that there exists $s > s^*$ such that D_s occurs:

$$\begin{aligned}
\Pr[\exists s > s^* | D_s] &\leq \sum_{s > s^*} \rho_s = \sum_{i=1}^{\infty} \rho_{s^*+i} \\
&\leq \sum_{i=1}^{\infty} e^{-\frac{\delta^2 \cdot p}{2} \cdot (s^* + i)} \\
&= \sum_{i=1}^{\infty} e^{-\frac{\delta^2 \cdot p}{2} \cdot s^*} \cdot e^{-\frac{\delta^2 \cdot p}{2} \cdot s^*} \\
&= e^{-\frac{\delta^2 \cdot p}{2} \cdot s^*} \sum_{i=1}^{\infty} e^{-\frac{\delta^2 \cdot p}{2} \cdot i}
\end{aligned}$$

Since for all $c > 0$ (and in particular for $c = \delta^2 p/2$), the series $\sum_{i=1}^{\infty} e^{-ci}$ converges to $\frac{1}{e^c - 1}$

$$= \frac{e^{-\frac{\delta^2 \cdot p}{2} \cdot s^*}}{e^{\frac{\delta^2 \cdot p}{2}} - 1}$$

Pick s^* such that $\Pr[\exists s > s^* | D_s] < 2^{-k}$, where k is the security parameter. Since for all $i \geq 1$ it holds that all honest parties agree on B at time $t + s^* + i$, then for all $i \geq 1$, by lemma 5.12 at time $t + s^* + i$ we must have

$$g_{t+s^*+i} \geq s^* + i - \sum_{j=1}^{s^*+i} T_{t+j}^{-x}$$

By property 2, $\sum_{j=0}^{s^*+i} T_{t+j}^{-x} < q(t + s^* + i - \text{layer}_B) + T_0 = q\Delta_t + q \cdot (s^* + i) + T_0$, hence

$$g_{t+s^*} \geq s^* + i - q\Delta_t - q \cdot s^* - q \cdot i - T_0 = (1 - q)(s^* + i) - q\Delta_t - T_0$$

Thus, for all $i > \frac{2}{1-q} \cdot (1 + q\Delta_t + T_0) - s^*$, B is in confident consensus.

□

5.4 Reduction to Graded Consensus

We show how to map the Spacemesh tortoise protocol to graded consensus with disruption tokens, such that every state of the tortoise protocol can be mapped to a corresponding graded consensus state and the state transitions satisfy properties 1 to 4 and 6 w.h.p.

To describe the mapping, we first need some notation. We denote $\text{tally}_t^{(P)}(B)$ the pair (v^+, v^-) of votes for and against B at time t, according to the view of party P. More explicitly, v^+ (resp. v^-) is the total weight of blocks in P's view at time t for which $\text{IsTortoiseValid}(B, t)$ returned 1 (resp. -1), whether or not it's confident. We denote $\text{margin}_t^{(P)}(B) = v^+ - v^-$ the *margin* of votes for B at time t.

We normalize grades using the average weight of a layer (in the current epoch). This is computed by taking the total weight of all ATXs published in the previous epoch, and dividing by the number of layers in an epoch. We denote this normalization factor for epoch i $E[W]_{(i)} \doteq \frac{\sum_{id \in activeset(i)} weight(id)}{\Delta_{\text{epoch}}}$. (We will occasionally abuse notation and allow, for a time t, the notation $E[W]_{(t)}$ to mean $E[W]_{(i)}$ where $i = \left\lfloor t/\Delta_{\text{epoch}} \right\rfloor$ is the epoch of t).

The state of a GCWDT at time t consists, for each party, of the following values:

- **opinion**: The opinion of a party P about B at time t is $\text{sign}(\text{margin}_t^{(P)}(B))$.
- **grade**: The grade of a party P about B at time t is $\frac{\left|\text{margin}_t^{(P)}(B)\right|}{\theta_L \cdot E[W]_{(t)}}$. Note that the graded opinion at time t is thus $\frac{\text{margin}_t^{(P)}(B)}{\theta_L \cdot E[W]_{(t)}}$.

  Intuitively, the grade measures how confident a party is in its opinion, in "θ_L units". Loosely, the size of the unit is set so that in a given layer two honest parties cannot disagree on the vote count by more than one unit unless the adversary spends more tokens than it received in that layer.
- **spent disruption tokens**: Let V_t^+ be the weight of positive (syntactically-valid) votes published in the interval $[t-1, t)$ by the adversary about B and $\overline{V_t^+}$ the weight of all positive votes *canceled* in that interval (i.e., positive votes published before time $t-1$ whose voting weight becomes 0 due to messages sent in the interval $[t-1, t)$). In the same way, define V_t^- as the weight of negative votes published in the interval, and $\overline{V_t^-}$ the weight of all negative votes canceled.

  Note that a vote cannot be counted as both published and canceled in the same interval; e.g., if the adversary publishes positive and negative votes using the same id (which would cause the voting weight of both to be considered 0 when they are both received), the weight would count once in V_t^+ (due to the publication of the positive weight), and once in V_t^-, but since these votes don't cancel *previously published* votes they won't be counted twice).

  We map the quantity $V_t^+ + \overline{V_t^-}$ to spending $T_t^+ = \frac{V_t^+ + \overline{V_t^-}}{\theta_L \cdot E[W]_{(t)}}$ positive disruption tokens, and $V_t^- + \overline{V_t^+}$ to spending $T_t^- = \frac{V_t^- + \overline{V_t^+}}{\theta_L \cdot E[W]_{(t)}}$ negative disruption tokens.

5.4.1 Mapping Special Layers

To define the mapping of real-world events to 'special layers' we will require the following technical lemma. Intuitively, it states that if we have a sequence of events that are guaranteed to occur w.p. at least p, even conditioned all other events in the sequence, then we can construct a set of completely independent events (using some auxiliary independent randomness) such that each event occurs w.p. exactly p. These constructed events will be our special layers.

Lemma 5.15. *Let $X_1, \ldots, X_n$ be a set of boolean random variables that satisfy: for every $i \in [n]$, for all $(x_1, \ldots, x_{i-1}) \in \{0,1\}^{i-1}$,*

$$\Pr\left[X_i = 1 | X_1 = x_1 \wedge \cdots X_{i-1} = x_{i-1}\right] \geq p \,.$$

Let $Z^{(1)}, \ldots, Z^{(n)}$ be a sequence of random variables such that $Z^{(i)} = \left\{Z_x^{(i)}\right\}_{x \in \{0,1\}^{i-1}}$ is a set of i.i.d variables uniformly distributed in $[0,1]$ and independent of $X_1, \ldots, X_n$ and of $Z^{(j)}$ for $j \neq i$. (We abuse notation and define $\{0,1\}^0 \doteq \{1\}$.)

Then there exists a set of i.i.d. Bernoulli variables $Y_1, \ldots, Y_n$ such that $Y_i = f(X_1, \ldots, X_i, Z^{(1)}, \ldots, Z^{(i)})$, $\Pr[Y_i = 1] = p$ and for all $i \in [n]$, $Y_i = 1 \Rightarrow X_i = 1$.

(We defer the proof to section 5.4.3.)

For every layer i, we define an event X_i that occurs iff the outcome of the weak coin is the same for all honest parties and either no honest opinion has grade at least 1 or the coin's value matches some honest opinion with grade at least 1. By the properties of the weak coin, we have that for all i, and any value $(x_1, \ldots, x_{i-1}) \in \{0,1\}^{i-1}$, it holds that

$$\Pr[X_i = 1 | X_1 = x_1 \wedge \cdots X_{i-1} = x_{i-1}] \geq p \,,$$

satisfying the conditions of lemma 5.15. Thus, we can construct a sequence of independent random variables $\{Y_i\}$ such that $\Pr[Y_i = 1] = p$ and $Y_i = 1 \Rightarrow X_i = 1$. Our mapping will define layer i to be special iff $Y_i = 1$.

5.4.2 Properties of our mapping

By our assumption about network synchrony, a block received by any honest party at time t will have been received by *all* honest parties at time $t+1$. This immediately leads to the following:

Claim 5.16. *For every two honest parties P and P' and every $x \in \{-,+\}$, it holds that*

$$\mathrm{tally}_{t+1}^{(P')}(B)[x] - \mathrm{tally}_t^{(P)}(B)[x] \geq -\overline{V_t^x} - \overline{V_{t+1}^x} \,.$$

Proof. At time $t+1$ party P' receives all blocks that party P received at time t. Thus, the only way the tally can decrease is if votes that were valid for P are canceled for P'. Since they were valid for P, the cancellation can't have been published before time $t-1$, and since they were canceled for P', they must have been published before $t+1$. Hence they are counted either in $\overline{V_t^x}$ or in $\overline{V_{t+1}^x}$. □

Honest parties always publish their blocks exactly on layer boundaries. Thus, honest votes at time $t-1$ will be included in $\mathrm{tally}_t^{(P)}(B)$ (and correspondingly in $\mathrm{margin}_t^{(P)}(B)$)) for every honest party P. In contrast, adversarial blocks can be published at any point in the interval $[t-1, t)$, and the adversary can control network scheduling. This means an adversarial block could be received by only some of the honest parties in the interval $[t-1, t)$. However, in this case network synchrony guarantees the rest of honest parties will receive the block in the interval $[t, t+1)$. This implies the following:

Claim 5.17. *Let X_t^+ and X_t^- be the weight of honest votes for (resp. against) B at time t. Then for every two honest parties P and P' and every $x \in \{-,+\}$, it holds that*

$$\mathrm{tally}_{t+1}^{(P')}(B)[x] - \mathrm{tally}_t^{(P)}(B)[x] \leq V_t^x + V_{t+1}^x + X_t^x \,.$$

Proof. By time $t+1$, P' will have received every block that P received by time t. Thus, its tally can only grow by the weight of votes that were received by P' and not by P; this includes the honest votes at time t, and potentially the adversarial votes in the interval $(t-1, t+1)$. □

Using claims 5.16 and 5.17, we can prove:

Lemma 5.18. *The GCWDT mapping satisfies property 1.*

Proof. Suppose honest party P has graded opinion xg at time t, and honest party P' has graded opinion $x'g'$ at time $t+1$. By our mapping, $xg \doteq \text{margin}_t^{(P)}(B)/\theta_L \cdot E[W]_{(t)}$ and $x'g' \doteq \text{margin}_{t+1}^{(P')}(B)/\theta_L \cdot E[W]_{(t)}$. Thus,

$$\begin{aligned} x(x'g' - xg) &= \frac{x\left(\text{margin}_{t+1}^{(P')}(B) - \text{margin}_t^{(P)}(B)\right)}{\theta_L \cdot E[W]} \\ &= \frac{\text{tally}_{t+1}^{(P')}(B)[x] - \text{tally}_t^{(P)}(B)[x] - (\text{tally}_{t+1}^{(P')}(B)[-x] - \text{tally}_t^{(P)}(B)[-x])}{\theta_L \cdot E[W]} \end{aligned}$$

by claim 5.16

$$\geq \frac{-\overline{V_t^x} - \overline{V_{t+1}^x} - (\text{tally}_{t+1}^{(P')}(B)[-x] - \text{tally}_t^{(P)}(B)[-x])}{\theta_L \cdot E[W]}$$

by claim 5.17

$$\begin{aligned} &\geq \frac{-\overline{V_t^x} - \overline{V_{t+1}^x} - (V_t^{-x} + V_{t+1}^{-x} + X_t^{-x})}{\theta_L \cdot E[W]} \\ &= -\frac{V_t^{-x} + \overline{V_t^x}}{\theta_L \cdot E[W]} - \frac{V_{t+1}^{-x} + \overline{V_{t+1}^x}}{\theta_L \cdot E[W]} - \frac{X_t^{-x}}{\theta_L \cdot E[W]} \end{aligned}$$

by the definition of T_t^x

$$= -T_t^{-x} - T_{t+1}^{-x} - \frac{X_t^{-x}}{\theta_L \cdot E[W]}$$

since the total honest weight is at most $(1-q)E[W]_{(t)}$

$$\geq -T_t^{-x} - T_{t+1}^{-x} - \frac{1-q}{\theta_L}$$

□

Claim 5.19. *The GCWDT mapping satisfies property 2.*

Proof. Our reduction maps two potential operations to spending a token: publication of a block and canceling the vote of a previously published block (by publishing a conflicting block or ATX). Define a *block slot* as a tuple (id, i) such that id is eligible to create a block at layer i. Note that for each block slot, the adversary can spend tokens in one of three ways:

- Publish a positive vote at time t_1, and then cancel the vote at time $t_2 > t_1$. In this case $V_{t_1}^+$ increases by the block weight as does $\overline{V_{t_2}^+}$.
- Publish a negative vote at time t_1, and then cancel the vote at time $t_2 > t_1$. In this case $V_{t_1}^-$ increases by the block weight as does $\overline{V_{t_2}^-}$.
- Publish both negative and positive votes at time t_1. In this case both $V_{t_1}^+$ and $V_{t_1}^-$ increase by the block weight.

In all three cases, a block with weight w contributes at most $w/(\theta_L \cdot E[W])$ to $\sum_{t=0}^{\infty} T_t^+$, and at most $w/(\theta_L \cdot E[W])$ to $\sum_{t=0}^{\infty} T_t^-$. Thus, the total number of positive (resp. negative) tokens that can be spent up to time t is bounded by the weight of blocks created in the interval $[\text{layer}_B, t]$. By lemma 5.5, this is bounded by $(1+\varepsilon)q(t - \text{layer}_B)$ except with negligible probability. □

Claim 5.20. *The GCWDT mapping satisfies property 3.*

Proof. Suppose all honest parties at time t have the same opinion x and grade at least $g_t \geq 1$. Assume w.l.o.g that $x = 1$ (the argument for $x = -1$ is symmetric). By our mapping, this means that for every honest party P, $\text{margin}_t^{(P)}(B) \geq \theta_L \cdot E[W]_{(t)}$.

Let x' and g' be the opinion and the grade, respectively, of an honest party at time $t+1$. By our mapping $x'g' = \text{margin}_{t+1}^{(P')}(B) / \theta_L \cdot E[W]_{(t)}$. Note that $\text{margin}_{t+1}^{(P')}(B)$ includes $\text{margin}_t^{(P')}(B)$, the votes of the honest miners at time $t+1$ and the disruption tokens spent at time $t+1$.

Because for every honest P, it holds that $\text{margin}_t^{(P)}(B) \geq \theta_L \cdot E[W]_{(t)}$, all honest miners vote x at time $t+1$ (due to the voting rules). Suppose that the weight of honest blocks at time $t+1$ is more than $\theta_L \cdot E[W]_{(t)}$, then the honest votes at time $t+1$ add 1 to the grade. We get that $xx'\text{margin}_{t+1}^{(P')}(B) \geq \text{margin}_t^{(P)}(B) + 1 - T_{t+1}^{-x}$.

It is left to show that each layer has more than θ_L honest miners Let L be the expected amount of blocks in a layer (this is a configuration parameter). We assume that blocks are distributed in the epoch's layer independently and uniformly at random. Given a layer i, let M be the number of blocks in that layer. By Chernoff bound, we have that $Pr[M \leq (1-\delta)L] \leq e^{-L\delta^2/2}$, for $0 < \delta < 1$.

We turn to calculate the probability that there are H honest blocks in layer i. Since the probability that a block is honest is $1-q$, the expected number of honest blocks in layer i is $M(1-q)$. By Chernoff bound we get that $Pr[H \leq (1-\delta')M(1-q)] \leq e^{-M(1-q)\delta'^2/2}$, for $0 < \delta' < 1$.

We need to compute the probability that $H < \theta_L$. Setting $(1-\delta')M(1-q) < \theta_L$, we get that $M < \theta_L/((1-\delta')(1-q))$. We have

$$Pr[\theta_L/((1-\delta')(1-q)) \leq (1-\delta)L] \leq e^{-L\delta^2/2}.$$

Hence,

$$Pr[H < \theta_L] = Pr[H \leq (1-\delta')M(1-q) \mid M < \theta_L/((1-\delta')(1-q))]Pr[\theta_L/((1-\delta')(1-q)) \leq (1-\delta)L] \leq e^{-L\delta^2/2}e^{-M(1-q)\delta'^2/2}.$$

□

5.4.3 Proof of lemma 5.15

We use the following claim in our proof:

Claim 5.21. *Let $p > 0$ and let X be a random variable with finite range R, and Y be a boolean random variable satisfying, for all $x \in R$, $\Pr[Y = 1 | X = x] \geq p$. Let $Z = \{Z_x\}_{x \in R}$ be a set of i.i.d. random variables uniformly distributed in $[0,1]$.*

If Z and (X, Y) are independent, then there exists a random variable $Y' = f(Y, X, Z)$ such that:

- *Y' and X are independent,*
- *$\Pr[Y' = 1] = p$ and*
- *$Y' = 1 \Rightarrow Y = 1$.*

Proof. For all $x \in R$, we define a boolean variable Y_x such that $Y_x = 1$ iff $Y = 1$ and $X = x$.

Denote $p_x = \Pr[Y_x = 1 | X = x]$. Since $\Pr[Y_x = 1 | X = x] = \Pr[Y = 1 | X = x]$, it follows that $p_x \geq p > 0$. We define $Y'_x = 1$ iff $Y_x = 1$ and $Z_x < p/p_x$. Then,

$$\begin{aligned}\Pr[Y'_x = 1 | X = x] &= \Pr[Y_x = 1 \wedge Z_x < p/p_x | X = x] \\ &= \Pr[Y_x = 1 \wedge Z_x < p/p_x \wedge X = x] / \Pr[X = x]\end{aligned}$$

Since Z_x is independent of X, Y_x:

$$= \Pr[Z_x < p/p_x] \cdot \Pr[Y_x = 1 \wedge X = x] / \Pr[X = x]$$
$$= \Pr[Z_x < p/p_x] \cdot \Pr[Y_x = 1 | X = x]$$
$$= \Pr[Z_x < p/p_x] \cdot p_x$$

Since Z_x is uniformly distributed in $[0, 1]$ and $p/p_x \leq 1$:

$$= (p/p_x) \cdot p_x = p$$

Set $Y' = \sum_{x \in R} Y'_x$. Note that Y' is a Boolean random variable, as given X, for all $x \neq X$ we have $Y'_x = 0$. By the law of total probability,

$$\begin{aligned}\Pr[Y' = 1] &= \sum_{x \in R} \Pr[X = x] \cdot \Pr[Y' = 1 | X = x] \\ &= \sum_{x \in R} \Pr[X = x] \Pr[Y'_x = 1 | X = x] \\ &= p \cdot \sum_{x \in R} \Pr[X = x] = p\end{aligned}$$

To see that Y' is independent of X, note that for all $x \in R$, it holds that

$$\Pr[Y' = 1 | X = x] = \Pr[Y'_x = 1 | X = x] = p = \Pr[Y' = 1]$$

Finally, by the construction of Y', if $Y' = 1$ it follows that $Y'_X = 1$, hence $Y_X = 1$ and therefore $Y = 1$. □

Proof of lemma 5.15. The proof is by induction on n. For $n = 1$, denote $X_0 = 1$ a constant random variable. The range of X_0 is $\{0, 1\}^0$. Since $\Pr[X_1 = 1 | X_0] = \Pr[X_1 = 1] \geq p$, by claim 5.21 it holds that there exists $Y_1 = f(X_1, X_0, Z^{(1)}) = f(X_1, Z^{(1)})$ such that $\Pr[Y_1 = 1] = p$.

Assume the induction hypothesis holds for j. Let $Y_1, \ldots, Y_j$ be the random variables guaranteed by the hypothesis for $X_1, \ldots, X_j$. Denote $X^* \doteq (X_1, \ldots, X_j)$ and $Z^* \doteq (Z^{(1)}, \ldots, Z^{(j)})$. By the conditions of the lemma, we have that $Z^{(j+1)}$ is independent of X^* and that for all $x^* \in \{0, 1\}^j$, it holds that $\Pr\left[X_{j+1} = 1 | X^* = x^*\right] \geq p$.

Thus, by claim 5.21, it holds that there exists $Y_{j+1} = f(X_{j+1}, X^*, Z^{(j+1)})$ that satisfies $\Pr\left[Y_{j+1} = 1\right] = p$, $Y_{j+1} = 1 \Rightarrow X_{j+1} = 1$ and Y_{j+1} is independent of X^*. Since $Y_1, \ldots, Y_j$ are a function of X^*, Z^*, Z^* is independent of $X_{j+1}, X^*, Z^{(j+1)}$ and Y_{j+1} is independent of X^* , it holds that Y_{j+1} is independent of $Y_1, \ldots, Y_n$. □

Claim 5.22. *The GCWDT mapping satisfies property 4.*

Proof. By our mapping, a special layer is a layer in which the outcome of the weak coin is the same for all honest parties and it matches an honest opinion with grade at least 1, if such opinion exists. By the properties of the weak coin, it is clear that specialness is independent for every layer. Moreover, since $1 - q$ of all blocks are honest, we get that the probability that a layer is special is at least $(1 - q)/2$, since if an honest opinion with grade at least 1 exists, the probability that the weak coin votes in the same way as the honest party is at least $1/2$. □

Claim 5.23. *The GCWDT mapping satisfies property 6.*

Proof. Suppose layer t is special. By definition, all honest miners have the same value of the weak coin, and if there is an honest miner with opinion x and grade at least 1, then the weak coin results in opinion x.

If the adversary spent disruption tokens at time t, the claim holds. Otherwise, by property 1, there are no honest parties with grade at least 1 and different opinions. Hence, all honest parties have the same opinion at time $t + 1$ (either all follow the weak coin, or those that don't follow the coin has the same opinion as those that follow the coin). □

5.5 Race-Freeness: All honest blocks are valid

Loosely, a protocol is "race-free" (as defined in [5]) if honest parties' rewards cannot be reduced by the actions of a malicious adversary (e.g., there cannot be a "race" to generate the next block, since in this case the actions of the adversary can cause the honest party to "lose" the race).

A complete proof of race-freeness is outside the scope of this paper, as it requires—in addition to the consensus protoocol—a full description of the incentive mechanism design. However, we *do* prove that there are no races to generate the next block: every honestly-generated block will be included in the final view. This property makes it much easier to design race-free incentive mechanisms for the Spacemesh protocol. Moreover, it immediately implies "optimal" chain quality and chain growth (see section 5.6).

Theorem 5.24. *Let $\theta_G < c(1-3q)/2$ for some $c \in (0,1)$. For every pair of parties P_1, P_2, time $i > 0$ and interval $t >$ hdist, if P_1 honestly generated a block B at time i, then the probability that P_2 does not consider B to be contextually valid at time $i + t$ is negligible in κ and t.*

Proof sketch. Since P_1 generated B honestly at time i, it must be syntactically valid, and the synchronicity of the network guarantees that all honest parties will receive it by layer $i + 1$.

Honest parties' input to the hare protocol is the set of syntactically-valid blocks they received "on time". Thus, all honest participants of the hare protocol for layer i will include B in their input, and the "Validity 1" property (cf. definition 6.1) of the hare protocols guarantees that its output will include B.

Denote ε_{hare} the probability that the hare protocol fails (or does not terminate) before layer $i+\text{hdist}/2$ (hdist is chosen so that this is negligible in κ; to simplify, think of hdist $= 1$—this is always possible to achieve by increasing the layer time, since the hare protocol termination time depends only on the network delay, not the layer time).

The protocol instructs honest parties to vote using the results of the hare protocol for the first hdist layers. If the hare protocol was successful (w.p. at least $1 - \varepsilon_{hare}$) then there are at least $\lceil \text{hdist}/2 \rceil$ consecutive layers such that every honest block votes for B while in the layers before that honest parties are explicitly neutral.

By lemma 5.5, the voting weight of adversarial blocks in the hdist consecutive layers starting at i is at most $c \cdot q \cdot T_{\text{ave}} \cdot \text{hdist}$ except with probability negligible in the security parameter, for any constant $c < 1$. (note that parameters such as ℓ and the average layer width would have to grow with the security parameter, and this dependence is worse the closer c is to 1).

Using a Chernoff bound, we can show that the probability that the voting weight of honest parties over $\lceil \text{hdist}/2 \rceil$ consecutive layers is less than $c \cdot \lceil \text{hdist}/2 \rceil \cdot (1-q)T_{\text{ave}}$ is negligible.

Hence, at layer $i + \text{hdist}$ the marginal weight of votes for B is at least

$$
\begin{aligned}
c \cdot \lceil \text{hdist}/2 \rceil \cdot (1-q)T_{\text{ave}} - c \cdot q \cdot T_{\text{ave}} \cdot \text{hdist} & \\
&\geq c \cdot (\text{hdist}/2) \cdot (1-q)T_{\text{ave}} - c \cdot q \cdot T_{\text{ave}} \cdot \text{hdist} \\
&= cT_{\text{ave}} \left((1-q)(\text{hdist}/2) - q \cdot \text{hdist}\right) \\
&= cT_{\text{ave}}\text{hdist} \cdot \left((1-q) - 2q\cdot\right)/2 \\
&= cT_{\text{ave}}\text{hdist} \cdot (1-3q)/2 \\
&\geq T_{\text{ave}}\text{hdist} \cdot \theta_G
\end{aligned}
$$

Thus, at time $i + t$, every honest party will consider B to be confidently valid; By mapping to the graded consensus model, we can use lemma 5.8 to conclude that every honest party will continue to consider B confidently valid forever. □

5.6 Putting it all together

Now that we have shown that the Spacemesh protocol satisfies the strong security properties above, we can use them to prove that it satisfies the definitions of section 3.1.

Theorem 5.25 (Chain Growth and Chain Quality)**.** *For every $c < 1$, if $(1-q) > 2q$ then there is a setting of parameters such that the Spacemesh protocol has chain growth rate $c \cdot T_{ave} \cdot (1-q)/$**layertime** and chain-quality $c \cdot (1-q)$*

Proof Sketch. By theorem 5.24, all honest blocks are considered valid. Since the eligibility of honest blocks is independent for different identities, we can use a Chernoff bound to show that the probability that there are less than $c \cdot T_{\text{ave}} \cdot (1-q)$ is negligible. Thus, in every layer—that is, **layertime** rounds, the chain will increase by at least $c \cdot T_{\text{ave}} \cdot (1-q)$, hence the protocol has a chain growth rate of $c \cdot T_{\text{ave}} \cdot (1-q)/$**layertime**.

Similarly, the probability that there are more than T_{ave}/c blocks in total in a layer is also negligible. Thus, the fraction of honestly-generated blocks in each layer is at least $c^2(1-q)$ (since we can take any $c < 1$, we can use $c' > 1$ as the parameter and set $c = \sqrt{c'}$). □

Theorem 5.26 (Consistency)**.** *If $(1-q) > 2q$ then there is a setting of parameters such that the Spacemesh protocol is consistent.*

Proof Sketch. First (by lemma 5.14), there exists some $z > 0$ such that every block is in confident consensus within z layers after it is published (lemma 5.14 relies on self-healing which requires $q \ll 1/3$; we can also use the Consistency property of the Hare protocol to show that all honest parties have the same initial opinion, and then apply lemma 5.8 to conclude that the grades will increase forever (and, in particular, no previously valid block will become invalid or vice versa).

Since honest parties reach consensus about every block within z layers, they will agree on the validity of all blocks that were published more than z layers in the past. Since the block order is a deterministic function of the valid blocks, all honest parties have a common prefix except for the last z layers. Moreover, by the the Validity 2 property of the Hare protocol, every "late" published block will be considered invalid by all honest parties; thus, no new valid blocks can be added to old layers, and consistency will be maintained between honest parties at different times. □

6 Hare Protocol

Definition 6.1 (Byzantine Agreement on Sets)**.** We say that a protocol Π run by parties $P_1, ..., P_n$, where P_i initially holds a set S_i, achieves *byzantine agreement on sets* if at the end of the protocol, every honest party outputs a set S_i' such that the following four conditions are satisfied:

- **Consistency:** Every honest party outputs the same same set S', i.e., if an honest party P_i terminates with set S_i' then $S_i' = S'$.
- **Validity 1:** If for every honest party P_i, $v \in S_i$, then $v \in S'$.
- **Validity 2:** If for every honest party P_i, $v \notin S_i$, then $v \notin S'$.
- **Termination:** All honest parties terminate the protocol with overwhelming probability.

Figure 6.1: BA protocol

Denote P_i's input set with S_i. P_i intitializes $k \leftarrow 0, k_i \leftarrow -1$.

- **Pre-Round:** (Executed only a single time)
  - **If P_i is active:** P_i broadcasts a message $(S_i, \mathit{preround})$ informing everybody of its set S_i.
  - By the end of this round, for each $v \in S_i$, if P_i received at least $f+1$ sets S_j s.t. $v \in S_j$, it forms from a certificate C_v containing the $f+1$ sets S_j. It then sets $C_i \leftarrow \bigcup_{v \in S_i} C_v$. It removes from S_i all values v for which it could not form a certificate C_v.
- **Round 0:**
  - **If P_i is active:** P_i broadcasts a message $(S_i, \mathit{status}, k, C_i, k_i)$ informing everybody of its current accepted set.
  - By the end of this round, P_i forms a safe value proof P for some set S.
- **Round 1:**
  - **If P_i is active:** P_i broadcasts a message $(S, \mathit{proposal}, k, P)$.
  - By the end of this round, if P_i received a proposal for set S from the leader, it sets $T_i = S$. Otherwise, it sets $T_i = \perp$.
- **Round 2:**
  - **If P_i is active:** If $T_i \neq \perp$, P_i broadcasts a message $(T_i, \mathit{commit}, k)$.
  - By the end of this round, if P_i received a proposal *from the leader* on a set T and $f+1$ commit messages for T, and no conflicting proposals from the leader on a set $T' \neq T$, it forms a commit certificate $\widetilde{C_i}$ for T containing these $f+1$ commit messages and sets $S_i = T$.
- **Round 3:**
  - **If P_i is active:** If P_i holds a commit certificate $\widetilde{C_i}$ for a proposed set T (in the current iteration), it broadcasts a message $(T, \mathit{notify}, k, \widetilde{C_i})$.
  - By the end of this round, if P_i received a notification message on a set T (along with commit certificate C), it sets $S_i = T, C_i = C$ and $k_i = k'$ iff C certifies (T, k') and $k' \geq k_i$.
- P_i sets $k \leftarrow k+1$ and continues from Round 1.

If at any point in the protocol, P_i party gathers $f+1$ notification headers (S, notify, k) from distinct parties on a set S, it terminates.

6.1 Overview

The protocol is iterative, with four rounds (Round 0 – Round 3) in an iteration, except for the first iteration which has an initial round, called "Pre-Round". If a party does not terminate by the end of Iteration j, where $j \geq 0$, it starts Iteration $j + 1$. The protocol is described in Figure 6.1. Section 6.2 analyzes the security of the protocol.

6.1.1 Role-Distribution Oracle O

In the protocol, a party can participate either *actively* or *passively*. To simplify the description of the protocol, we suppose that the parties can query a special *role-distribution oracle* O during any given round. The oracle O tells a party P_i whether it is active or passive for the current round, and also provides P_i with a proof it can use to convince other parties that it indeed holds the role in question.

We assume that any party P_i may query O to learn its role, but may not query O to learn P_j's role, where $j \neq i$. More formally, for round r, if P_i queries $O(j, r)$, the oracle O returns either $\perp$ in case $j \neq i$, or a tuple $(role_{i,r}, \pi_{i,r})$, where $role_{i,r}$ denotes the role of P_i in round r and $\pi_{i,r}$ is a proof for $role_{i,r}$ that can be verified by any party. Honest parties send messages only in rounds in which they are active. Thus, each party has to include its role proof with each message it sends.

In addition to being active or passive, a party may hold the role of the *leader* within a given iteration. The leader may only be chosen from the active parties in Round 1. As opposed to above, a party may not query the oracle to know if it is the leader. Instead, once active parties (in Round 1) prove their role during Round 1, each party (active or passive) identifies the leader from their respective proofs of activity. All honest parties choose the same leader as long as they have seen the same proofs by the end of Round 1. If different parties see different sets of proofs, then they may elect a different leader. This does not harm the protocol. Note that a party can not locally tell whether it is actually the leader until it sees all the proofs of active parties.

In practice, O will be instantiated using the *unique signature* approach by Micali et al. [**?**].

6.1.2 Safe Value Proof for set S

In the following, k is called *iteration* and k_i *certified iteration*.

For any iteration $k \geq 0$, a safe value proof P for a set S consists of either:

- – $f + 1$ status messages for iteration k including sets $S_0, ..., S_f$ that form the set $S \doteq \bigcup_i S_i$ and have $k_i = -1$.
  - – $|S|$ sets $\mathcal{F}_v$: $\forall v \in S$, a family $\mathcal{F}_v$ of $f + 1$ sets from *preround* messages that contain v, i.e., $\mathcal{F}_v := \{S_v \mid v \in S_v\}$ and $|\mathcal{F}_v| = f + 1$.
- – $f+1$ status messages for iteration k such that at least one of them contains the set S with certified iteration of at least 0. Moreover, S must be claimed by a status message with the *highest* certified iteration among all these status messages. In other words, $P = (s_1, ..., s_{f+1}, C)$ where $s_j = (S_j, status, k, C_j, k_j)$. If $k_j = \mathsf{max}(k_1, ..., k_{f+1}) \geq 0$, then C certifies (S_j, k_j).

6.2 Security Analysis

In the following we assume that at most f parties are not honest and at least $f + 1$ parties are honest. In particular, f is less than half of all active parties (in each round), so the probability that the leader – who is elected uniformly at random – is honest is at least 0.5. We assume that once an honest party saw a message, all honest parties see that message by a length of a round.

Lemma 6.2. *Let P_i be any honest party that broadcasts a notify message on a set S in Round 3 of iteration k. If C is a commit certificate for (S', k') where $k' \geq k$, then $S' = S$.*

Proof. We prove the statement by induction on k'.

- **Base Case:** $k' = k$**.** Suppose that C certifies (S', k) and $S' \neq S$. C includes $f+1$ commit messages. Since there are at most f malicious parties, at least one of these messages must have been sent by an honest party P_j. Therefore, by the end of Round 1, party P_j received a valid proposal from the leader of iteration k on S'. In turn, by our assumption, all honest parties have seen this proposal by the end of Round 2. Since $S' \neq S$, party P_i does not notify on S (since it detects two proposals from the leader), a contradiction.

- **Induction Hypothesis.** Suppose the statement holds for $k' \geq k$.

- **Induction Step:** $k' + 1$**.** By the end of iteration k, every honest party P_j (that has not terminated) must receive P_i's notify message. By the base case, no certificate can certify (S', k) for $S' \neq S$. Thus, at the end of Round 3 in iteration k, party P_j must set $S_j = S$. Moreover, by the induction hypothesis, no certificate can certify (S', k') for $S' \neq S$. Thus, at the end of Round 3 in iteration k', party P_j still has $S_j = S$.

  Therefore, any honest party that is active at the beginning of iteration $k'+1$ sends a status message for the set S. It follows that the leader in this iteration cannot gather $f + 1$ status messages on a set $S' \neq S$. Hence, no honest party sends a commit message on S' in this iteration. Therefore, it is not possible to form a certificate on a set $S' \neq S$ in iteration $k' + 1$ that includes $f + 1$ commit messages on the set S'.

□

Corollary 6.3. *Suppose party P_i is the first honest party to broadcasts a notify message. If P_i notifies on S in iteration k and C certifies (S', k') where $k' \geq k$, then $S' = S$.*

Lemma 6.4. *The protocol in fig. 6.1 satisfies consistency.*

Proof. Suppose that P_i is the first honest party to terminate, and its output is S. Hence, P_i have seen a commit certificate on S. Let P_j be any honest party that terminates. Thus, P_j must have seen some commit certificate. Applying lemma 6.2, this certificate must certify S. Therefore, P_j outputs S. □

Lemma 6.5. *The protocol in fig. 6.1 satisfies Validity 1.*

Proof. Let v be some value such that $v \in S_i$ for all honest parties P_i at the beginning of Pre-Round. Then, at the beginning of Round 0 (of Iteration 0) the same holds true. Indeed, in Pre-Round each active honest party sends a set that includes v, thus by the end of this round, every honest party P_i obtains at least $f + 1$ sets that include v, and so does not discard v from S_i. Thus, in Round 1 at most f status messages with sets that do not include v are sent.

Consider a set S where $v \notin S$. Suppose that P is a safe value proof for S. P includes $f + 1$ status messages from distinct parties with sets $S'_1, ..., S'_{f+1}$, none of which include v, a contradiction. Hence, there could not be a (valid) proposal on S in Iteration 0.

As a result, in Round 2 of Iteration 0 no party sends a commit message on S, and therefore no notify message in Round 3 can be on S. Hence, no honest party can terminate with the output S in Iteration 0. It follows that no honest party P_i sets $S_i = S$ at the end of that iteration.

Inductively, with an argument similar to lemma 6.2, in subsequent iterations no honest party P_i sets $S_i = S$, and so there could not be a notify message on S. Hence, no honest party can terminate with the output S. □

Lemma 6.6. *The protocol in fig. 6.1 satisfies Validity 2.*

Proof. The proof is very similar to the proof of lemma 6.5. Let v be some value such that $v \notin S_i$ for all honest parties P_i at the beginning of Pre-Round. Thus, there are at most f preround messages with sets that include v. Consider a set S where $v \in S$. Suppose that P is a safe value proof for S. P includes $f + 1$ preround messages from distinct parties with sets $S'_1, ..., S'_{f+1}$, each includes v, a contradiction. Hence, there could not be a (valid) proposal on S in Iteration 0.

As a result, in Round 2 of Iteration 0 no party sends a commit message on S, and therefore no notify message in Round 3 can be on S. Hence, no honest party can terminate with the output S in Iteration 0. It follows that no honest party P_i sets $S_i = S$ at the end of that iteration. Now, we can inductively apply a similar argument to ensure that in no subsequent iteration P_i sets $S_i = S$, hence no honest party can terminate with the output S. □

Lemma 6.7. *The protocol in fig. 6.1 is expected to terminate in at most 9 rounds.*

Proof. First note that since the leader is elected after sending a proposal message (that is, no one knows who the leader is before the proposals are sent), the adversary cannot prevent (with a noticeable probability) the proposal of an honest leader from propagating in the network and eventually reach all honest parties.

Since an honest leader is elected with probability at least $1/2$, we can compute the expected number of iterations until an honest leader is elected as $\sum_{i=1}^{\infty} \frac{i}{2^i} = 2$. We claim that if an honest party is elected leader in an iteration, then the protocol terminates by the end of that iteration for every honest party.

We first show that a party can always form a safe value proof, thus when an honest party is elected as leader, they can always propose, and have a proof for, a (possible empty) set. We analyze two subcases:

- No honest party has updated its set S_i after Pre-Round ended. In this case, every honest party holds a certificate of inclusion for every value in S_i. Each party is therefore ensured to obtain at least $f + 1$ status messages with proofs of inclusion for every value in the respective sets. It can form from them a safe value proof that includes a proof of inclusion for every value in the union of these $f + 1$ sets.

- At least one honest party has updated its set S_i after Pre-Round ended. The corresponding certificate of that party has a certified iteration of at least 0. In this case, a safe value proof may include only status messages. Since there are at least $f + 1$ active honest parties in Round 0, each party can gather the sufficient amount of messages to form a safe value proof.

To finish the proof it remains to argue that if the leader is honest, all honest parties terminate by the end of that iteration. This claim directly follows from the properties of our network assumption: all honest parties see the proposal by the beginning of Round 1, hence $f + 1$ commit messages by honest parties are sent at the beginning of Round 2, and subsequently $f + 1$ notify messages by honest parties are sent at the beginning of Round 3, and received by all honest parties by the end of that round.

Finally, since an honest leader is expected to be elected by the second iteration, all parties are expected to terminate within at most 9 rounds, as claimed.

□

6.3 Communication Optimized Protocol

The protocol in the previous section is not efficient in terms of communication complexity (CC). The main problem is Round 1 of the protocol (during at least the first iteration), in which every party has to send a signed set per value in its own set. This results in a cost of roughly $O(n^3)$. To overcome this limitation, we can rely on the properties of the gossip network. Concretely, an honest party P_i does not need to send anything beyond its accepted set in Round 1 if $k_i = -1$. This follows from the fact that if P_i in Round 1 sees any set that it thinks was sent in Round 0, then it can be sure that an honest leader will have received the same set by the beginning of Round 2. Therefore, an honest leader can exhaustively search all the *preround* sets that it received up to this point to form a valid SVP without the need of parties sending the expensive certificates of their accepted sets in Round 1. This reduces the CC to $O(n^2)$. It is important to note that the CC here is measured in the number of bits that are put onto the gossip network in total by all parties throughout the protocol. The standard of the literature treats all the messages that are sent over the gossip network as different messages. This includes messages that are being echoed, for example in the form of certificates. However, there is no need to treat all

of these messages as distinct messages at the *network* level. Namely, the gossip network removes the need of echoing messages on the protocol level, since a message sent by a party is always echoed at the network level by any other party, until everybody eventually sees the message. To sum up, echoing messages on the protocol level when using a gossip network blows up the communication complexity by an unnecessary factor of n, whereas using its special diffusion properties in a non-blackbox fashion is much more efficient.

References

[1] I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi. Communication complexity of byzantine agreement, revisited. arXiv, 2017. URL: http://arxiv.org/abs/1704.02397v2.

[2] H. Abusalah, J. Alwen, B. Cohen, D. Khilko, K. Pietrzak, and L. Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 357–379. Springer, 2017.

[3] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[4] J. Benet, N. Greco, et al. Filecoin: A decentralized storage network, 2017. https://filecoin.io/filecoin.pdf.

[5] I. Bentov, P. Hubácek, T. Moran, and A. Nadler. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies. *IACR Cryptology ePrint Archive*, 2017:300, 2017. URL: http://eprint.iacr.org/2017/300.

[6] A. Biryukov and D. Khovratovich. Egalitarian computing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 315–326, Austin, TX, 2016. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/biryukov.

[7] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography*, TCC'13, pages 315–333, Berlin, Heidelberg, 2013. Springer-Verlag. URL: http://dx.doi.org/10.1007/978-3-642-36594-2_18, doi:10.1007/978-3-642-36594-2_18.

[8] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[9] X. Boyen, C. Carr, and T. Haines. Blockchain-free cryptocurrencies. a rational framework for truly decentralised fast transactions. Cryptology ePrint Archive, Report 2016/871, 2016. http://eprint.iacr.org/2016/871.

[10] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos. Algorand agreement super fast and partition resilient byzantine agreement. Technical report, 2018.

[11] B. Cohen and K. Pietrzak. The chia network blockchain, 2019. https://www.chia.net/assets/ChiaGreenPaper.pdf.

[12] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *NSDI*, 2016.

[13] I. Eyal and E. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography*, 2014.

[14] B. Fisch, J. Bonneau, N. Greco, and J. Benet. Scaling proof-of-replication for filecoin mining, 2018. https://web.stanford.edu/~bfisch/porep_short.pdf.

[15] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015. http://eprint.iacr.org/2014/765.

[16] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*, pages 291–323. Springer, 2017. doi:10.1007/978-3-319-63688-7_10.

[17] R. Gennaro, C. Gentry, B. Parno, and M. R. 0001. Quadratic span programs and succinct nizks without pcps. In T. J. 0001 and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[18] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017. doi:10.1145/3132747.3132757.

[19] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short pcps. In *Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity*, CCC '07, pages 278–291, Washington, DC, USA, 2007. IEEE Computer Society. URL: https://doi.org/10.1109/CCC.2007.10, doi:10.1109/CCC.2007.10.

[20] A. Kiayias and G. Panagiotakos. On trees, chains and fast transactions in the blockchain, 2016. URL: http://eprint.iacr.org/2016/545.

[21] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *International Conference on Autonomous Agents and Multiagent Systems*, 2015.

[22] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In *Financial Cryptography and Data Security*, pages 528–547, 2015.

[23] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao. Scaling nakamoto consensus to thousands of transactions per second, 2018. URL: https://arxiv.org/abs/1805.03870.

[24] T. Moran and I. Orlov. Proofs of space-time and rational proofs of storage. *IACR Cryptology ePrint Archive*, 2016:35, 2016. URL: http://eprint.iacr.org/2016/035.

[25] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Bitcoin.org*, 2008. URL: http://www.bitcoin.org/bitcoin.pdf.

[26] N. Narula. Cryptographic vulnerabilities in iota, 2017. https://medium.com/@neha/cryptographic-vulnerabilities-in-iota-9a6a9ddc4367.

[27] R. Pass, L. Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt 2017*, 2017. URL: http://eprint.iacr.org/2016/454.

[28] R. Pass and E. Shi. Fruitchains: A fair blockchain. In *PODC 2017*, 2017. URL: http://eprint.iacr.org/2016/916.

[29] S. Popov. The tangle, 2017. https://iota.org/IOTA_Whitepaper.pdf.

[30] M. Rauchs, A. Blandin, and A. Dek. Cambridge bitcoin electricity consumption index, 2019. https://www.cbeci.org/comparisons/.

[31] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in Bitcoin. In *Financial Cryptography*, 2016.

[32] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. Spectre: A fast and scalable cryptocurrency protocol, 2016. https://eprint.iacr.org/2016/1159.

[33] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *19th Financial Cryptography and Data Security*, 2015.

[34] Y. Sompolinsky and A. Zohar. PHANTOM: A scalable blockdag protocol. *IACR Cryptology ePrint Archive*, 2018:104, 2018. URL: http://eprint.iacr.org/2018/104.

[35] F. Voight. p2pool: Decentralized, dos-resistant, hop-proof pool, 2011. https://bitcointalk.org/index.php?topic=18313.0.

[36] E. Wall. Iota is centralized, 2017. https://medium.com/@ercwl/iota-is-centralized-6289246e7b4d.

[37] H. Yu, I. Nikolic, R. Hou, and P. Saxena. Ohie: Blockchain scaling made simple, 2018. URL: https://arxiv.org/abs/1811.12628.