# Combining GHOST and Casper

Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, Yan X Zhang

October 31, 2019

We present "Gasper," a formal proof-of-stake-based consensus protocol, which is an idealized version of the proposed Ethereum 2.0 beacon chain. The protocol combines Casper FFG, a finality tool, with LMD GHOST, a fork-choice rule. We prove plausible liveness, probabilistic liveness under suitable synchrony assumptions, and safety under both static and dynamic validator set assumptions.

## Contents

# 1. Introduction

Our goal is to create a consensus protocol for a proof-of-stake blockchain. This paper is motivated by the need to construct Ethereum's "beacon chain" in its sharding design. However, the construction is general enough that it could be used to e.g. design the sole blockchain in some brand-new protocol with no sharding involved.

The consensus protocol presented in this paper combines the Casper FFG (the Friendly Finality Gadget) finalization rule and the Latest Message Driven (LMD) Greediest Heaviest Observed SubTree (GHOST) fork-choice rule. Casper FFG [7] is a "finality gadget," an algorithm that marks certain blocks in a blockchain as *finalized* so that participants can have full confidence that the block is considered to be part of the chain's history. It is designed to work on top of either proof-of-work or proof-of-stake chains. LMD GHOST is a *fork-choice rule* where *validators* (participants) *attest* to blocks to signal support for those blocks, an idea which is similar to voting. Our protocol combines the two ideas for a full proof-of-stake blockchain, where time is split into *epochs* that create "checkpoints" at *epoch boundary blocks*, and participants build new blocks and attest to blocks with LMD GHOST on top of finalized blocks.

In Sections 2 and 3, we define our primitives, provide background knowledge, and state our goals, In Section 4, we give our main protocol Gasper. In Sections 5, 6, and 7, we formally prove Gasper's desired qualities. In Section 8, we summarize some differences between Gasper and the actual Ethereum specifications, such as delays in accepting attestations, delaying finalization, and dynamic validator sets. We conclude with some thoughts and ideas for future research in Section 9.

# 2. Setup and Goals

The main goal of this paper is to describe and prove properties of a proof-of-stake blockchain with certain safety and liveness claims. In this section, we give some general background on consensus protocols and blockchain. As the literature uses a diverse lexicon, the primary purpose of this section are to clearly pick out a specific set of vocabulary to be reused for the rest of the paper.

## 2.1. Consensus Protocols, Validators, Blockchain

Our goal is to create a *consensus protocol* (or *protocol* for short from this point on), which is an agreed suite of algorithms for a set of entities (nodes, people, etc.) to follow in order to obtain a consensus history of their state, even if the network is unreliable and/or if many validators are malicious.

We call the entities (people, programs, etc.) who participate in protocol *validators*, denoted by the set $\mathcal{V}$. They are called "validators" because of their data-validation role in the Ethereum 2.0 beacon chain, though the specifics of this role are not part of our abstract protocol. Other works use many different words for the same concept: e.g. "replicas" in classic PBFT literature, "block producers" in DPoS and EOS, "peers" on Peercoin and Nxt, or "bakers" on

Tezos. The validators are connected to each other on a (peer-to-peer) *network*, which means they can broadcast *messages* (basically, packets of data) to each other. In the types of protocols we are interested in, the primary types of messages can be interpreted as proposing pieces of data called *blocks*. The first block is called the *genesis block* and acts as the initial "blank slate" state. Other blocks are descriptions of state transitions with a pointer to a *parent* block[1]. A *blockchain* is just an instantiation (i.e. with specific network conditions, validators, etc.) of such a consensus protocol where we use blocks to build the overall state.

We do not want all blocks seen on the network to be accepted as common history because blocks can conflict with each other. These conflicts can happen for honest reasons such as network latency, or for malicious reasons such as byzantine validators wanting to double-spend money. Graph-theoretically, one can think of a choice of history as a choice of a "chain" going from the genesis block to a particular block. Thus, a consensus history is a consensus of which chain of blocks to accept as correct. This is why we call such an instantiation of our protocol a "blockchain."

**Example 2.1.** Assume the goal of a protocol is to keep a ledger of every validator's account balances. Then their collective state is the ledger, and each block captures a state transition, such as a monetary transaction from one validator to another (e.g., "Yunice sends coin with ID 538 to Bureaugard,"). One type of conflict we may want to avoid is having both the aforementioned transaction and another transaction with content "Yunice sends coin with ID 538 to Carol" be accepted into consensus history, an act commonly called *double-spending*. The current consensus state can be determined by starting from a "genesis state" and sequentially processing each message in the consensus history (hopefully one that all honest validators will agree with), starting from the genesis block.

## 2.2. Messages and Views

In our model, a validator $V$ interacts with the network by broadcasting *messages*, which are strings in some language. When an honest validator $V$ broadcasts a message $M$, we assume $M$ is sent to all validators on the network[2].

The main type of message *proposes* a *block*, which is a piece of data, to the network. Other messages can be bookkeeping notices such as voting for blocks ("attestation"), putting new validators on the blockchain ("activation"), proving bad actions of other validators ("slashing"), etc. depending on the specific protocol. We assume that each message is digitally signed by a single validator,

[1] Our choice that each block has a unique parent is almost taken for granted in blockchains; one can imagine blockchains where blocks do not need to have parents, or can have more than one parent. See, for instance, Hashgraph [2]. Even the very general structure of "block" itself may be limiting!

[2] We model the validators as sending a single message to an abstract "network," which handles the message. In practice, it may be part of the protocol that e.g., every honest validator rebroadcasts all messages that they see, for robustness. We consider these implementation details not important to our paper.

which means we can accurately trace the author of each message and attacks such as impersonation of honest validators are not possible.

Due to network latency and dishonest validators (delaying messages or relaying incorrect information), validators may have different states of knowledge of the full set of messages given to the network; we formalize this by saying that a validator either *sees* or *does not see* each message given to the network at any given time.

Each message may have one or more *dependencies*, where each dependency is another message. At any time, we *accept* a message if and only if all of its dependencies (possibly none) are accepted, defined recursively. We now define the *view* of a validator $V$ at a given time $T$, denoted as $\mathsf{view}(V, T)$, as the set of all the **accepted** messages the validator has seen so far.

We also have a "God's-eye-view" that we call the *network view*, defined to be the set of accepted messages for a hypothetical validator that has seen (with no latency) all messages any validator has broadcast at any time (this includes messages sent by a malicious validator to only a subset of the network). We will treat the network as a "virtual validator", so we use $\mathsf{view}(\mathsf{NW}, T)$ to denote the network view at time $T$. For any validator $V$ and any given time $T$, $\mathsf{view}(\mathsf{NW}, T)$ includes all the messages in $\mathsf{view}(V, T)$, though the timestamps may be mismatched.

**Example 2.2.** At time $T$, suppose validator $V$ sees the message $M$ with content "Yunice sends coin number 5340 to Bob," but has not yet seen the message $M'$ containing "Yunice obtains coin number 5340" (which is on the network but has not yet made its way to $V$). Also suppose that in our protocol $M$ depends on $M'$ (and no other message), which captures the semantic meaning that we need to obtain a coin before spending it, and that $V$ cannot and should not act on $M$ without seeing $M'$. In this situation we say that $V$ has seen but has not accepted $M$, so $M \notin \mathsf{view}(V, T)$. However, the network has seen both; so if the network accepts $M'$, we have $M \in \mathsf{view}(\mathsf{NW}, T)$.

We now make some assumptions on the structure of views:

- Everyone (the validators and the network) starts with a single message (with no signature) corresponding to a single agreed-upon *genesis block*, denoted $B_{\text{genesis}}$, with no dependencies (so it starts out automatically accepted into all views).

- Each block besides $B_{\text{genesis}}$ refers to (and depends on) a *parent block* as part of its data. Thus, we can visualize $\mathsf{view}(V)$ as a directed acyclic graph (in fact, a directed tree) rooted at $B_{\text{genesis}}$, with an edge $B \leftarrow B'$ if $B$ is the parent of $B'$, in which case we say $B'$ is a *child* of $B$. We call these *parent-child* edges to differentiate from other types of edges.

- A *chain* in such a protocol would then be a sequence of pairwise parent-child edges $B_1 \leftarrow B_2 \leftarrow \cdots$ If there is a chain from $B$ to $B'$, we say $B'$ is a *descendant* of $B$. We say $B$ is an *ancestor* of $B'$ if and only if $B'$ is a descendant of $B$. We say two blocks $B$ and $B'$ *conflict* if they do not equal and neither is a descendant of the other.

- The above setup implies each block $B$ uniquely defines a (backwards) chain starting from $B_{\text{genesis}}$ to $B$, and this must be the same chain in any view that includes $B$ (thus we do not need to include a view as part of its definition). We call this *the chain of* $B$, or $\mathsf{chain}(B)$.

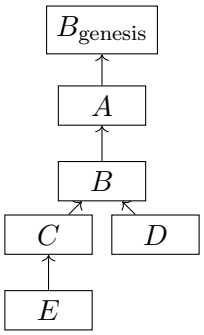



Figure 1: A graph visualization of a view (only visualizing blocks and not other messages) that occurs in the types of protocols we care about.

**Example 2.3.** See Figure 1 for an example of a view. Arrows portray parent-child edges. Views look like trees rooted at a single genesis block $B_{\text{genesis}}$. Here blocks $C$ and $D$ conflict (as well as $E$ and $D$). $E$ has the longest chain, which is $\mathsf{chain}(E) = (B_{\text{genesis}}, A, B, C, E)$.

There are some technicalities to our wording in this section that motivates our definition of *views* but are not very important to the core of this paper. To not distract the reader, we defer these details to Appendix A.

## 2.3. Proof-of-stake

The first and most influential approach to blockchain protocols is Bitcoin [16], using a proof-of-work model. This approach makes it computationally difficult to propose blocks, using the simple and intuitive fork-choice rule, "the block with the heaviest chain is the head of the chain." Miners (the analogy of validators) simply propose blocks to build on the heaviest chain (the chain with the most amount of computational work), which does not mathematically guarantee any non-genesis block as "correct" but offers a probabilistic guarantee since it becomes less and less likely to conflict with a block once other miners build on top of it. Then the consensus history is simply the chain with the heaviest amount of work, which ideally keeps growing. The elegance of proof-of-work is paid for by the expensive cost of computational work / electricity / etc. to propose a block.

In this paper, we want to create a *proof-of-stake* blockchain, where a validator's voting power is proportional to their bonded stake (or money) in the system. Instead of using computational power to propose blocks, proposing blocks is essentially free. In exchange, we need an additional layer of mathematical theory

to prevent perverse incentives that arise when we make proposing blocks "easy." We now shift our paper's focus to a family of blockchain designs with proof-of-stake in mind.

To start, we assume we have a set of $N$ *validators* $\mathcal{V} = \{V_1, \ldots, V_N\}$ and that each validator $V \in \mathcal{V}$ has an amount of *stake* $w(V)$, a positive real number describing an amount of collateral. We make the further assumption that the average amount of stake for each validator is 1 unit, so the sum of the total stake is $N$. All of our operations involving stake will be linear, so this scaling does not change the situation and makes the bookkeeping easier. We assume validator sets and stake sizes are fixed for the sake of focusing on the theoretical essentials of the protocol. We address issues that occur when we relax these conditions in practical use in later sections, especially Section 8.

The main ingredients we need are:

- A *fork-choice rule*: a function $\mathsf{fork}()$ that, when given a view $G$, identifies a single leaf block (a vertex with no descendants) $B$. This choice produces a unique chain $\mathsf{fork}(G) = \mathsf{chain}(B)$ from $B_{\text{genesis}}$ to $B$ called the *canonical chain*. The block $B$ is called the *head of the chain* in view $G$. Intuitively, a fork-choice rule gives a validator a "law" to follow to decide what the "right" block should be. For example, the *longest chain rule* is a fork-choice rule that returns the leaf block which is farthest from the genesis block. This rule is similar with the "heaviest chain rule" used by Bitcoin.

- A concept of *finality*: formally, a deterministic function $F$ that, when given a view $G$, returns a set $F(G)$ of *finalized* blocks. Intuitively, finalized blocks are "blocks that everyone will eventually think of as part of the consensus history" or "what the blockchain is sure of."

- *Slashing conditions*: these are conditions that honest validators would never violate and violating validators can be provably caught, with the idea that violators' stake would be *slashed*, or destroyed. Slashing conditions incentivize validators to follow the protocol (the protocol can reward honest validators for catching dishonest validators violating the conditions, for example).

The key concept we use to define these ingredients are *attestations*, which are votes (embedded in messages) for which blocks "should" be the head of the chain (using the fork-choice rule), backed up by deposited stake. To prevent validators from double-voting or voting in protocol-breaking ways, we enforce *slashing conditions* which can be used to destroy a validator's stake, incentivizing validators to follow the protocol. One can see an idealized version of the protocol we will present in Figure 2; in Section 4 we will define all the details.

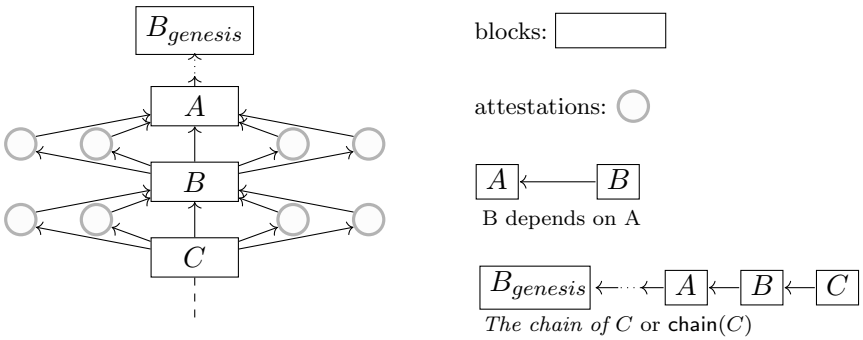



Figure 2: An ideal situation where in each slot, a new block is created with the parent block being the last block created, and validators in the corresponding committee all perfectly attest to the new block, etc. In less idealized situations, the blocks may fork, we may have missing attestations, etc.

## 2.4. Byzantine Validators, PBFT

In a protocol, we call a validator *honest* if he/she follows the protocol, and *byzantine* otherwise. We typically assume strictly less than $p = \frac{1}{3}$ of validators are byzantine. This constant can be traced to Practical Byzantine Fault Tolerance (PBFT) from [8], a classic consensus protocol in byzantine tolerance literature; PBFT ensures that the system runs correctly as long as less than $\frac{1}{3}$ of the *replicas* (synonymous with our *validators*) are byzantine, as proven in [3]. This constant $\frac{1}{3}$ tolerance appears frequently in many works based on PBFT (the main idea is that a pigeonhole principle argument needs the 1/3 to work, such as in our proof in Section 5), with Casper FFG [7] being the most directly relevant to our work. We will show details of Casper FFG in Section 3.1.

One of the most important aspects of PBFT is that it works under *asynchronous* conditions, which means we have no bounds on how long it may take for messages to be received. Thus, it is robust to the demands of blockchain / cryptocurrencies, where we often worry about adversarial contexts. We will discuss synchrony assumptions further in Section 2.6.

## 2.5. Safety and Liveness

The ultimate goal for the honest validators is to grow a finalized chain where all blocks form "logically consistent" state transitions with each other (despite having validators potentially go offline, suffer latency problems, or maliciously propose conflicting state changes). Formally, this translates to two desired properties:

**Definition 2.4.** We say a consensus protocol has:

- *safety*, if the set of finalized blocks $F(G)$ for any view $G$ can never contain two conflicting blocks. A consequence of having safety is that any validator

view $G(V)$'s finalized blocks $F(G(V))$ can be "completed" into a unique subchain of $F(G(\mathsf{NW}))$ that starts at the genesis block and ends at the last finalized block, which we call the *finalized chain*.

- *liveness*, if the set of finalized blocks can actually grow. There are different ways to define liveness. For our paper, we say the protocol has:
  - *plausible liveness*, if regardless of any previous events (attacks, latency, etc.) it is always possible for new blocks to be finalized (alternatively, it is impossible to become "deadlocked"). This is to prevent situations where honest validators cannot continue unless someone forfeits their own stake.
  - *probabilistic liveness*, if regardless of any previous events, it is probable for new blocks to be finalized (once we make some probabilistic assumptions about the network latency, capabilities of attackers, etc.).

  On first glance, the latter implies the former, but the situation is more subtle; the former is a purely non-probabilistic property about the logic of the protocol; the latter requires (potentially very strong) assumptions about the context of the implementation to guarantee that the protocol "usually works as intended." We discuss this contrast further in Section 7.3.

Our main goal in this paper is to prove these vital properties for our main protocol in Section 4.

**Remark** (Syntax versus Semantics)**.** It should not be immediately obvious that chains have anything to do with logical consistence of state transitions, and this requires more work on the part of the protocol designer to ensure. For example, one can dictate a protocol where a block that allows a user to spend a coin $S$ forbids an ancestor block from using $S$ in another transaction. In this situation, if Xander obtained a coin $S$ from block $B_x$ and writes two blocks $B_y$ and $B_z$, such that $B_y$'s data includes Xander paying $S$ to Yeezus and $B_z$'s data includes Xander paying $S$ to Zachariah, then the logical idea that these actions are inconsistent corresponds to the graph-theoretical property that $B_y$ and $B_z$ conflict as blocks. The language of chains and conflicting blocks is enough to serve any such logic, so we assume in our paper that the syntax of what blocks can be children of what other blocks has already been designed to embed the semantics of logical consistency, which allows us to ignore logic and only think in terms of chains and conflicting blocks.

## 2.6. Time, Epochs, and Synchrony

In our model, time progresses in *slots* (e.g., imagine every slot is 6 seconds long). A collection of $C$ slots is called an *epoch*, where $C$ is a fixed global constant[3]. The *epoch* $j$ of slot $i$ is $\mathsf{ep}(i) = j = \lfloor \frac{i}{C} \rfloor$. In other words, the blocks belonging to epoch $j$ have slot numbers $jC + k$ as $k$ runs through $\{0, 1, \ldots, C-1\}$. The genesis block $B_{\text{genesis}}$ has slot number 0 and is then the first block of epoch 0.

[3]To give a sense of scale, $C = 64$ in [11].

The main purpose of epochs is to divide time into pieces, the boundaries between which can be thought of as "checkpoints." This allows concepts from Casper FFG to be used, as we will see in Section 3.1.

We should not assume the network is guaranteed to have the same thing happen for everyone, or even that different validators have the same view of time. The study of consensus protocols address this issue by having different *synchrony conditions*, such as:

- a *synchronous* system has explicit upper bounds for time needed to send messages between nodes;
- an *asynchronous* system has no guarantees; recall that PBFT [8] works under asynchrony.
- a *partially synchronous* system can mean one of two things depending on context: (i) explicit upper bounds for delays exist but are not known a priori; (ii) explicit upper bounds are known to exist after a certain unknown time $T$.

The work [12] establishes bounds on fault tolerance in different fault and synchrony models, focusing on partial synchrony.

To capture the notion (ii) of partial synchrony above, we say that the network is $t$-*synchronous* at time $T$ (where $T$ and $t$ are both in units of slots) if all messages with timestamps at or before time $(T-t)$ are in the views of all validators at time $T$; e.g., if each slot is 6 seconds, then $(1/2)$-synchrony means that all messages are received up to 3 seconds later. We will make this assumption when we look at probable liveness, and make no assumptions about synchrony otherwise unless explicitly stated.

**Remark.** It is possible to e.g., receive messages "from the future." Suppose Alexis sends a message timestamped at 00:01:30 PT and Bob receives it 1 second later with a clock that is 3 seconds behind Alexis's. Bob sees this 00:01:30 PT message on his own clock timestamped at 00:01:28 PT because he is a net total of 2 seconds behind. To make analysis easier, we can assume that all (honest) validators always delay the receiving of a message (i.e., do not add it into their view) until their own timestamps hit the message's timestamp. This allows us to assume no messages are read (again, by honest validators) in earlier slots than they should.

# 3. Main Ingredients

## 3.1. Casper FFG

In [7], Buterin and Griffith introduce Casper FFG (the Friendly Finality Gadget), a tool that defines the concepts of *justification* and *finalization* inspired by PBFT literature, on top of a blockchain (proof-of-stake, proof-of-work, or otherwise) with tree-like structures as we have in our setup.

- Every block has a *height* defined by their distance from the genesis block (which has height 0). Equivalently, the height of $B$ is the length of $\mathsf{chain}(B) - 1$.

- We define *checkpoint* blocks to be blocks whose height is a multiple of a constant $H$ (in [7], $H = 100$). We define the *checkpoint height* $h(B)$ for a checkpoint block $B$ as the height of $B$ divided by $H$, which is always an integer. Thus, we can think of the subset of checkpoints in the view as a subtree, containing only blocks whose heights are multiples of $H$.

- *Attestations* are signed messages containing "checkpoint edges" $A \to B$, where $A$ and $B$ are checkpoint blocks. We can think of each such attestation as a "vote" to move from block $A$ to $B$. The choices of $A$ and $B$ depend on the underlying blockchain and is not a part of Casper. Each attestation has a *weight*, which is the stake of the validator writing the attestation. Note that ideally $h(B) = h(A)+1$, but this is not a requirement. For example, if $H = 100$, it is possible for an honest validator to somehow miss block 200, in which case their underlying blockchain may want them to send an attestation from checkpoint block 100 to checkpoint block 300.

**Remark.** Note that in our setup, we have time set up in units of *slots*, which are grouped into *epochs*. This is analogous to (but not exactly identical to) block heights and checkpoint blocks. In Section 4.1 we discuss this further.

Casper FFG also introduces the concepts of *justification* and *finalization*, which are analogous to phase-based concepts in the PBFT literature such as *prepare* and *commit* (see e.g. [8]):

- In each view $G$, there is a set of *justified* checkpoint blocks $J(G)$ and a subset $F(G) \subset J(G)$ of *finalized* checkpoint blocks.

- In a view $G$, a checkpoint block $B$ is *justified* (by a checkpoint block $A$) if there are attestations voting for $A \to B$ with total weight at least $2/3$ of total validator stake. Equivalently, we say that there is a *supermajority link* $A \overset{\mathsf{J}}{\to} B$. This is a view-dependent condition, because the view in question may have or have not seen all the relevant attestations to break the $2/3$ threshold (or even having seen the block $B$ itself), which is why the set of justified blocks is parametrized by $G$.

- In a view $G$, if $A \in J(G)$ (equivalently, $A$ is justified), and $A \overset{\mathsf{J}}{\to} B$ is a supermajority link with $h(B) = h(A) + 1$, then we say that $A \in F(G)$ (equivalently, $A$ is *finalized*).

Finally, Casper introduces *slashing conditions*, which are assumptions that we make about honest validators. When they are broken by a validator $V$, a different validator $W$ can *slash* $V$ (destroy $V$'s stake and possibly getting some sort of "slashing reward") by offering proof that $V$ violated the conditions.

**Definition 3.1.** The following *slashing conditions*, when broken, cause a validator violating them to have their stake slashed:

- (S1) No validator makes two distinct attestations $\alpha_1$ and $\alpha_2$ corresponding to checkpoint edges $s_1 \to t_1$ and $s_2 \to t_2$ respectively with $h(t_1) = h(t_2)$.
- (S2) No validator makes two distinct attestations $\alpha_1$ and $\alpha_2$ corresponding respectively to checkpoint edges $s_1 \to t_1$ and $s_2 \to t_2$, such that

$$h(s_1) < h(s_2) < h(t_2) < h(t_1).$$

As Casper is a finality gadget and not a complete protocol, it assumes the underlying protocol has its own fork-choice rule, and at every epoch all the validators run the fork-choice rule at some point to make one (and only one) attestation. It is assumed that honest validators following the underlying protocol will never be slashed. For example, if the protocol asks to make exactly one attestation per epoch, then honest validators will never violate (S1).

The main theorems in Casper are (slightly paraphrased):

**Theorem 3.2** (Accountable Safety)**.** Two checkpoints on different branches cannot both be finalized, unless a set of validators owning stake above some total provably violated the protocol (and thus can be held accountable).

**Theorem 3.3** (Plausible Liveness)**.** It is always possible for new checkpoints to become finalized, provided that new blocks can be created by the underlying blockchain.

**Remark.** Jain et al. [14] observed and provided some solutions to the limitations of Casper FFG in [7] and the Proof-of-Stake model. Palmskog et al. [17] provided a mechanical proof assistant for Casper FFG's accountable safety and plausible liveness in the Coq Proof Assistant. The authors modified the blockchain model in Coq, Toychain, to use for the Casper FFG with Ethereum's beacon chain specs in order to see how Casper FFG's proofs will work in practice.

## 3.2. LMD GHOST Fork-Choice rule

In this section, we introduce the Latest Message Driven Greediest Heaviest Observed SubTree (LMD GHOST) rule, the most basic version of our fork-choice rule (recall honest validators follow this algorithm to determine the canonical chain). In Section 4 we will modify the rule to satisfy our other purposes.

The Greediest Heaviest Observed SubTree rule (GHOST) is a fork-choice rule introduced by Sompolinsky and Zohar [20]. Intuitively, GHOST is a greedy algorithm that grows the blockchain on sub-branches with the "most activity." It is a flexible fork-choice rule that can e.g. be used for either a proof-of-work or a proof-of-stake blockchain. See Figure 3 for a graphical visualization.

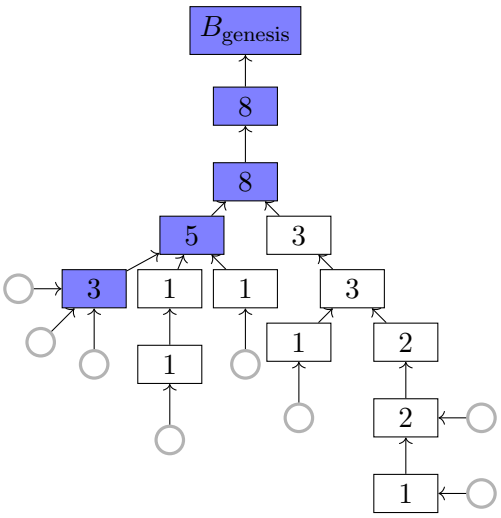



Figure 3: An example of the LMD-GHOST fork-choice rule. The number in each block $B$ is the weight (by stake), with all attestations (circles) having weight 1 in our example. A validator using this view will conclude the blue chain to be the canonical chain, and output the latest blue block with weight 3 to be the head of the chain.

LMD GHOST is a modified version of GHOST algorithm adapted by Ethereum for their beacon chain. We need the following definition:

**Definition 3.4.** Given a view $G$, Let $M = [M_1 \dots M_N]$ be the latest attestation message of each validator. The *weight* $w(G, B, M)$ is defined to be the sum of the stake of the validators $i$ whose last attestation $M_i$ is to $B$ or descendants of $B$.

The idea of LMD GHOST is that at any fork, we use the weights of the subtrees created by the fork as a heuristic and assume the subtree with the heaviest weight is the "right" one, as evident from the name of the algorithm. See Algorithm 3.1 for a formal description.

**Algorithm 3.1** LMD GHOST Fork Choice Rule.

**procedure** LMD-GHOST($F$)
  $B \leftarrow B_{\text{genesis}}$
  Let $M = [M_1 \dots M_N]$ be the most-recent attestations of the $N$ validators
  **while** $B$ has children **do**
    $B \leftarrow \underset{B' \text{ child of } B}{\arg\max}\ w(G, B', M)$
    (ties are broken by hash of the block header)
  **return** B

# 4. Main Protocol: Gasper

We now define our main protocol Gasper, which is a combination of the GHOST and Casper FFG ideas. The main concepts we will need are:

- *(Epoch boundary) pairs* of a chain: given a chain, certain blocks are picked out, ideally one per epoch, to play the role of Casper's *checkpoints*. However, a block may appear more than once as a checkpoint on the same chain (this is a nuance not found in Casper; we expound on this in Section 4.1), so we use ordered pairs $(B, j)$, where $B$ is a block and $j$ is an epoch, to disambiguate. These will be called *epoch boundary pairs*, or *pairs* for short.
- *Committees*: in each epoch, the validators are partitioned into *committees*, one per slot. In each slot, one validator from the designated committee proposes a block and all the members of that committee will attest to what they see as the head of the chain (which is hopefully the block just proposed) with the fork-choice rule HLMD GHOST (a slight variation of LMD GHOST).
- *Justification and Finalization*: these concepts are virtually identical to that of Casper, except that instead of justifying and finalizing checkpoint blocks, we justify and finalize pairs (formally: given a view $G$, $J(G)$ and $F(G)$ are sets of pairs instead).

## 4.1. Epoch Boundary Blocks and Pairs

Recall any particular block $B$ uniquely determines a chain, $\mathsf{chain}(B)$. Define $\mathsf{EBB}(B, j)$, the $j$-th *epoch boundary block* of $B$, to be the block with the highest slot less than or equal to $jC$ for every epoch $j$. Let the latest such block be $\mathsf{LEBB}(B)$, or the *last epoch boundary block* (of $B$). For every block $B$, $\mathsf{EBB}(B, 0) = B_{\text{genesis}}$. For an example, see Figure 4.

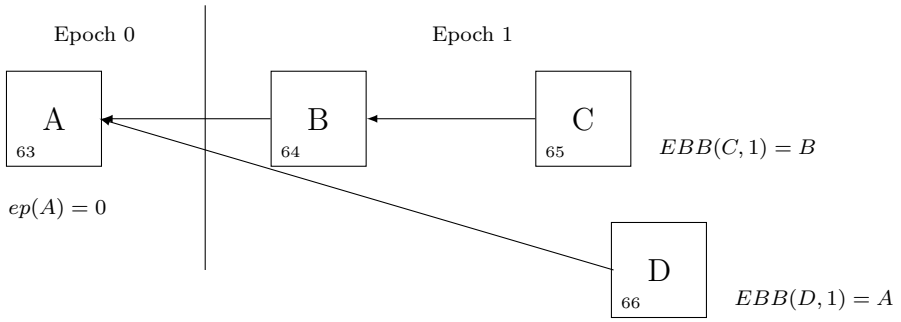



Figure 4: Suppose blocks $A \leftarrow B \leftarrow C$ form a chain at slots $63, 64, 65$ respectively, and blocks $A \leftarrow D$ form a chain where $\mathsf{slot}(D) = 66$. Then $\mathsf{EBB}(C, 1) = B$ and $\mathsf{EBB}(D, 1) = A$. Observe even though both blocks are 1-st epoch boundary blocks, $\mathsf{ep}(A) = 0$.

Note that an epoch boundary block $B'$ may not be an epoch boundary block of all chains that include it, unless $\mathsf{slot}(B') = jC$ for some epoch $j$. Furthermore, a block may be the epoch boundary block of possibly multiple chains at different epochs.

To disambiguate, justification and finalization will be done on *epoch boundary pairs* (or *pairs* for short) instead of blocks. Given a pair $P = (B, j)$, we say that $P$ has *attestation epoch* $j$, using the notation $\mathsf{aep}(P) = j$.

**Remark.** We briefly address why we use pairs instead of checkpoint blocks. Casper FFG is a "finality gadget," meaning it is designed to place a layer of finality on top of a blockchain which has probabilistic liveness (so likely to be live), which gives a steady new supply of checkpoint blocks. For our design, probabilistic liveness is not a priori assumed, so we need to take into account worst-case scenarios where we may be put into a state where we have not seen a new block for a while.

As an example, we may e.g. have the block $B$ be the last block from epoch 1 in our canonical chain, but be currently at epoch 3 with no new block on that chain. In the original Casper FFG, we are expecting probabilistic liveness. Then in the analysis we would need to differentiate the idea of "checkpoint in epoch 2" and "checkpoint in epoch 3" even though the "best" block we have for both is $B$, which is why we use $(B, 2)$ and $(B, 3)$ to represent the two distinct ideas separately.

Another reason for this choice is that Casper FFG makes no assumptions about time of the underlying blockchain – only block heights are important, with no notion of *slots* or *epochs*. The use of block height is a natural choice in proof-of-work blockchains due to the Poisson process of mining new blocks serving somewhat as a system clock, with blocks coming at fairly irregular times. Gasper, as a proof-of-stake protocol, can have blocks coming in at controlled regular intervals as part of the protocol (instead of depending on a random process), so instead the notion of time is explicitly desired in the protocol. To capture this notion of time, each object in our blockchain then naturally requires the knowledge of both the data (captured in a block) and the time (captured in the epoch count), which leads naturally to the idea of pairs.

## 4.2. Committees

The point of committees is to split up the responsibilities among the validators. To start, assume the validators have access to a sequence of random length-$N$ permutations $\rho_0, \rho_1, \ldots$, as functions $\{1, 2, \ldots, N\} \to \{1, 2, \ldots, N\}$. In the scope of this article, we assume that we get these random permutations from a random oracle[4].

Recall that time is split into epochs, of $C$ slots each. Each permutation $\rho_j$ will be used only during epoch $j$. Its role is to pseudorandomly select validators

[4] Achieving non-gameable randomness on the blockchain is itself an interesting problem, stimulating research such as verifiable delay functions; however, in this article we will take this randomness for granted.

into $C$ *committees*, each of whom has responsibilities for one slot of the epoch. To be precise:

- During epoch $j$, we would like to split the set of validators $V$ into $C$ equal-size committees $S_0, S_1, \ldots, S_{C-1}$ (we assume $C|N$ for easier notation; dealing with "roughly equal" size committees does not change the essence of our approach).
- Therefore, we define for each $k \in \{0, 1, \ldots, C-1\}$, $S_k$ consists of all of the $N/C$ validators of the form $\mathcal{V}_{\rho_j(s)}$, where $s \equiv k \pmod{C}$. Note that, for an epoch $j$, the sets $S_0, S_1, \ldots, S_{C-1}$ partition the entire set of validators $\{V_1, \ldots, V_N\}$ across all the slots of epoch $j$, as desired.

To summarize, $\rho_j$ first shuffles the validators, then places them in committees based on their original index modulo $C$.

## 4.3. Protocol for Each Slot

### 4.3.1. Blocks and Attestations

Now, in each slot, the protocol dictates 2 types of "committee work" for the committee assigned to that slot: one person in the committee needs to *propose* a new block, and everyone in the committee needs to *attest* to the heads of the chain (in their own views). Both responsibilities follow the same fork-choice rule, which is a variation of LMD GHOST.

For a view $G$, we use $\mathsf{HLMD}()$ to denote the Hybrid LMD GHOST fork-choice rule, a slight variation of the LMD GHOST fork-choice rule, defined as follows (note it relies on the definition of $J(G)$, the justified pairs in $G$, which we will define later in this section):

**Algorithm 4.1** Hybrid LMD GHOST Fork Choice Rule

**procedure** $\mathsf{HLMD}(G)$
  $(B_J, j) \leftarrow$ the justified pair with highest attestation epoch $j$ in $J(G)$
  $B \leftarrow B_J$
  Let $M = [M_1 \ldots M_n]$ be the most-recent messages
  **while** $B$ has children **do**
    $B \leftarrow \underset{B' \text{ child of } B}{\arg\max}\ w(G, B', M)$
    (ties are broken by hash of the block header)
  **return** B

The responsibilities of the protocol during slot $i = jC+k$, where $k \in \{0, 1, \ldots, C-1\}$ are (all mentions of time are computed from the point-of-view of the validator's local clock, which we assume to be synced within some delta):

1. At the beginning of the slot $i$ ($i = jC + k$), validator $V = V_{\rho_j(k)}$ (i.e., the first member of the committee $S_k$ of epoch $j$) is designated as the *proposer* for that slot. The proposer computes $\mathsf{HLMD}(\mathsf{view}(V, i)) = B_P$, the canonical head of the chain in his/her view, then *proposes a block* $B$, which is a message containing:

a) $\mathsf{slot}(B) = i$, the slot number.

b) $P(B) = B_P$, a pointer to the parent block; in other words, we always build a block on top of the head of the chain.

c) $\mathsf{newattests}(B)$, a set of pointers to all the *attestations* (to be defined next) $V$ has accepted, but have not been included in any $\mathsf{newattests}(B')$ for a block $B'$ that is an ancestor of $B$.

d) Some implementation-specific data (for example, "Yunice paid 4.2 ETH to Brad" if we are tracking coins), the semantics of which is irrelevant for us.

For dependencies, $B$ depends on $P(B)$ and all attestations in $\mathsf{newattests}(B)$. (so for example, if we see a block on the network but do not see its parent, we ignore the block until we see its parent)

2. At time $(i + 1/2)$, each validator $V$ in committee $S_k$ computes $B' = \mathsf{HLMD}(\mathsf{view}(V, i+1/2))$, and publishes an *attestation* $\alpha$, which is a message containing:

   a) $\mathsf{slot}(\alpha) = jC + k$, the slot in which the attester is making the attestation. We will also use $\mathsf{ep}(\alpha)$ as shorthand for $\mathsf{ep}(\mathsf{slot}(\alpha))$.

   b) $\mathsf{block}(\alpha) = B'$. We say that $\alpha$ *attests to* $\mathsf{block}(\alpha)$. We will have $\mathsf{slot}(\mathsf{block}(\alpha)) \leq \mathsf{slot}(\alpha)$, and "usually" get equality by quickly attesting to the block that was just proposed in the slot. The intuition here is that $\alpha$ is considered a vote for $B'$ for HLMD GHOST purposes.

   c) A *checkpoint edge* $\mathsf{LJ}(\alpha) \xrightarrow{\mathsf{V}} \mathsf{LE}(\alpha)$. Here, $\mathsf{LJ}(\alpha)$ and $\mathsf{LE}(\alpha)$ are epoch boundary pairs in $\mathsf{view}(V, i + 1/2)$. We define them properly in Section 4.4.2. The intuition here is that this is a "Casper FFG vote" for the transition between the two epoch boundary pairs, much like transitions between two Casper checkpoint blocks.

   For dependencies, $\alpha$ depends on $\mathsf{block}(\alpha)$. So we ignore an attestation until the block it is attesting to is accepted into our view (this is one of the bigger differences between theory and implementation; we discuss this more in in Section 8).

Both proposing blocks and publishing attestations mean immediately adding the corresponding message to their own view and broadcasting to the network. Recall that the messages proposing (non-genesis) blocks and publishing attestations have digital signatures. Two ways to interpret the attestation consideration delay: 1) only add attestations from 1 slot ago to a view, 2) add the attestations to a view as soon as it is received, but don't consider the attestation while running HLMD ghost until it's at least 1 slot old. Not sure which one is better? YW I think right now it is not good (unless EF really wants to) to add all this delay stuff into the pure protocol, because then it corrupts every single type of argument because the delayed nature may become relevant. These thigns are worth thinking about, but a bad design choice for a math paper to have it in the main protocol. The main protocol should be something that is reasonable to analyze with the brain that captures the major points of the protocol. YZ

An honest validator following HLMD to cast their vote may violate slashing condition in some extreme situations. A patch is proposed to fix the problem. When running the fork choice, validator should filter out branches that do not contain correct latest justified in head state. Not sure if we should address this in our protocol. [YW] [I think something like this may be worth mentioning as an implementation detail in practice vs/theory, but the theory is already a bit too distracting with details that does not help the reader get a good mental picture.][YZ]

#### 4.3.2. Attestation Consideration Delay

Consider moving the relevant part of attestation inclusion delay here from Practice vs Theory? Is it better if we make Attestation Consideration Delay part of our protocol/theory?[YW] [I don't think it should be here; it really confuses the main parts of the protocol.][YZ]

#### 4.3.3. Attestation Inclusion Delay

Consider moving the Attestation Inclusion Delay here? It looks more relevant here? Can we just ignore Attestation Inclusion Delay in the protocol and treat it as an implementation detail?[YW]

I think there are two types of Attestation Inclusion Delay. The first type is the delay added by the designer to promote decentralization, which is same as the one we have in practice vs theory. The second type is caused by network latency or proposer inefficiency in collecting all attestations. If EF increases the slot time into 12 seconds, they may not need to add Attestation Inclusion Delay in the actual implementation. But I am not sure if Attestation Inclusion Delay due to other reasons can be ignored in pure Gasper. [YW] [Still currently leaning on leaving them as addons instead of the pure protocol.][YZ] @01/01 Our definitions of justification and finalization are both based on the assumption that there is no attestation inclusion delay. So is the probabilistic liveness proof for justification liveness and finalization liveness. It's critical how we deal with the Attestation Inclusion Delay. Is an attestation accepted when it is received or when it is included in the chain? Let's talk about this one when you have time.[YW]

### 4.4. Protocol for Each Epoch

#### 4.4.1. Processing Justification and finalization

I want to add specific rules about how to update the last justified pair and last pair. I got stuck here, so I propose three options we may use to process justification and finalization.[YW]

Option 1: keep what we currently have in the protocol. The validator updates the last justified pair and last pair in his/her view constantly. Not sure if this will work in all scenarios.[YW]

Option 2: In each epoch, all validators run the Hybrid LMD GHOST fork-choice rule at 1/2 time of the first slot of the epoch, update the justified pairs and

finalized pairs based on their view in that slot, considering the FFG votes from attestations that have been included in the canonical chain. Depending on the network latency, and percentage of honest validators, all attestations needed to justify a block may not get included in the chain in the first slot of the following epoch, and it may take 2 or 3 epochs (maybe more) for the attestations to get included in the canonical chain. Do we treat this as an implementation detail or do we cover this in our protocol? YW

Option 3: We ignore the detail that attestations need to be included in the canonical chain to get counted. At 1/2 time of the first slot of every epoch, validators consider the FFG vote which is at least 1 slot old in their view to update the justified pair and finalized pair in their view. With this option, we assume all attestations get included in the canonical chain in a timely manner given certain synchrony condition and enough incentive for proposers to include attestations in their blocks. YW

My opinion: Option 2 and option 3 make no difference for our safety argument, but they will result in different liveness argument. I think option 3 aligns with our liveness argument where we have 1/2 sychrony. If we have to use option 1, we might need to revise the liveness part a little bit. YW

### 4.4.2. Justification

**Definition 4.1.** Given a block $B$, we define[5] $\mathsf{view}(B)$, the *view of* $B$, to be the view consisting of $B$ and all its ancestors in the dependency graph. In other words, $\mathsf{view}(B)$ is the smallest set of information required to accept $B$. Any view that includes the block $B$ can compute $\mathsf{view}(B)$ independently and obtain the same result.

Recall that an attestation $\alpha$ for Gasper contains what we called a "FFG vote" between two epoch boundary pairs. We now define them:

1. First, let $B = \mathsf{LEBB}(\mathsf{block}(\alpha))$.
2. We define $\mathsf{LJ}(\alpha)$, the *last justified pair of* $\alpha$, to be the last justified pair[6] in $J(\mathsf{view}(B))$.
3. We define $\mathsf{LE}(\alpha)$, the *last epoch boundary pair of* $\alpha$, to be $(B, \mathsf{ep}(\mathsf{slot}(\alpha)))$.

We say that the attestation $\alpha$ gives a *checkpoint edge* $\mathsf{LJ}(\alpha) \xrightarrow{\mathsf{V}} \mathsf{LE}(\alpha)$. In this situation, while $\alpha$ *attests to* $\mathsf{block}(\alpha)$, we say that $\alpha$ *checkpoint-attests to* $\mathsf{LE}(\alpha)$.

**Definition 4.2.** We say that there is a *supermajority link* from pair $(A, j')$ to pair $(B, j)$ if the attestations with checkpoint edge $(A, j') \xrightarrow{\mathsf{V}} (B, j)$ have total stake more than $\frac{2N}{3}$. In this case, we write $(A, j') \xrightarrow{\mathsf{J}} (B, j)$.

[5] We have tyrannically overworked the notation $\mathsf{view}()$ by this point, but there should be no ambiguity when we know the type of its parameter.

[6] Note that this is the last justified pair in $\mathsf{view}(B)$, which is **not** necessarily the same as the last justified pair of the view of the validator during the attestation. The reason for this design choice is to make the guarantee that everyone who sees $B$ as the correct last epoch boundary block should get the same source block as well, irrespective of their view at the time making the attestation.

Given a view $G$, a block $B$, and an epoch $j$, we say that the pair $(B, j)$ is *justified in* $G$, or equivalently that $B$ *is justified in* $G$ *during epoch* $j$, if:

- $B = B_{\text{genesis}}$, $j = 0$; or
- for some justified pair $(A, j')$, $(A, j') \overset{\mathsf{J}}{\to} (B, j)$.

We use $J(G)$ to denote the set of justified pairs in $G$.

Everything here is analogous to Casper FFG. Inside the chain of $\mathsf{block}(\alpha)$ is a sub-chain created by the epoch boundary blocks of that chain, starting from $B_{\text{genesis}}$ and ending at $B$. We want to focus on this subchain of blocks (represented by pairs to allow for boundary cases) and justify epoch boundary pairs with many attestations. An attestation $\alpha$ checkpoint-attesting to $B$ during epoch $j$ can be thought of as a vote to transition from some last justified pair $(B', j')$ to the new pair $(B, j)$, visualized as the checkpoint edge $(A, j') \to (B, j)$, voting for $(B, j)$ to be the next justified pair. If $2N/3$ stake worth of attestations agree, we create a supermajority link $(A, j') \to (B, j)$ and justify $(B, j)$.

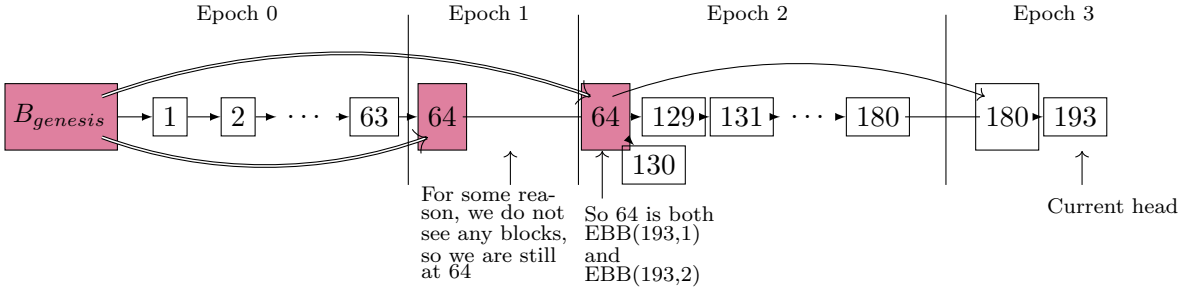



Figure 5: Blocks in red are justified, with double edges corresponding to supermajority links (in $\mathsf{view}(180)$). The single arc edge is the checkpoint edge for a current attestation to block 193.

**Example 4.3.** In Figure 5, we show an example of all of these concepts. The context is that a committee member runs $\mathsf{HLMD}(G)$ on her view $G$ to obtain block 193, which is her head of the chain. She is then supposed to attest to 193 with an attestation $\alpha$.

On $\mathsf{chain}(193)$ (pictured in the figure), $\mathsf{LE}(\alpha) = (180, 3)$, even though $\mathsf{ep}(180) = 2$, because our attestation has epoch 3, and we were looking for $3 * 64 = 192$ but did not see it (one can imagine, like in our figure, that we "pull up" block 180 to show that it is $\mathsf{EBB}(193, 3)$). In $\mathsf{view}(180)$, the last justified (by epoch number, not slot) pair is $(64, 2)$; (given our context, this is not automatic from the picture, because even though $G$ may see the attestations that justified 64, it may be possible that in $\mathsf{view}(180)$ we did not see enough attestations to justify 64) this means $\alpha$ corresponds to a checkpoint edge $(64, 2) \to (180, 3)$.

### 4.4.3. Finalization

Given our notion of justification and a new fork-choice rule, we are now ready to define the notion of finalization for pairs. Finalization is a stronger notion of justification in the sense that the moment one view sees $(B, j)$ as finalized for some $j$, no view will finalize a pair conflicting with $(B, j)$ unless at least validators with $N/3$ stake commit slashable offences.

**Definition 4.4.** For a view $G$, we say that $(B_0, j)$ is *finalized*[7] if there is an integer $k \geq 1$ and blocks $B_1, \ldots, B_k$ such that:

- $(B_0, j), (B_1, j+1), \ldots, (B_k, j+k)$ are adjacent epoch boundary pairs in $\mathsf{chain}(B_k)$;
- $(B_0, j), (B_1, j+1), \ldots, (B_k, j+k)$ are all in $J(G)$;
- $(B_0, j) \xrightarrow{\mathsf{J}} (B_k, j+k)$.

We define $F(G)$ to be the set of finalized pairs in the view $G$; we also say that a block $B$ is *finalized* if $(B, j) \in F(G)$ for some epoch $j$.

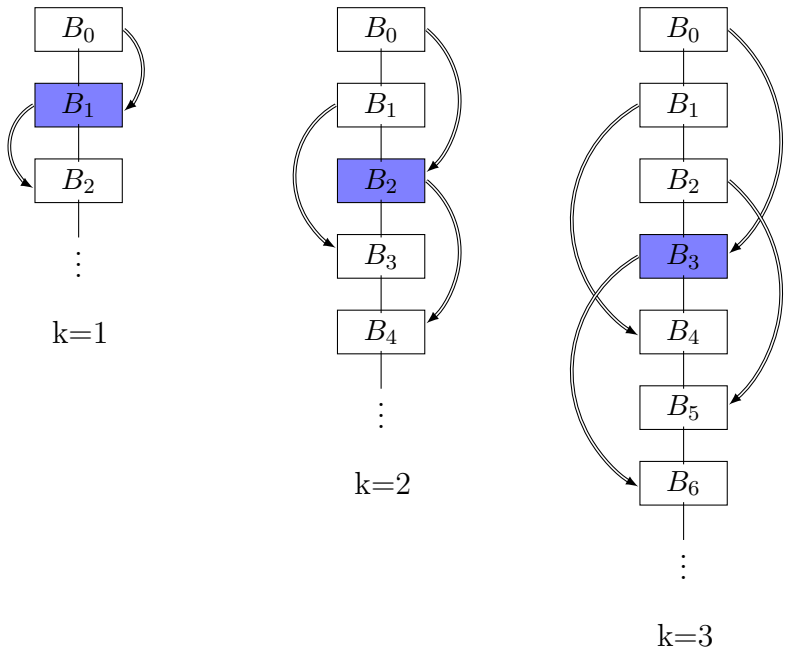



Figure 6: Illustrative examples of Definition 4.4 for $k = 1, 2, 3$, respectively.

[7] As comparison, in [7], a justified block $B_1$ is *finalized* if there is a supermajority link from $B_1$ to $B_2$, where $B_1$ and $B_2$ are adjacent epoch boundary blocks.

**Remark.** In Figure 6, we see some examples of finalization. On the left, we have $k = 1$, which is what we expect to happen in the vast majority of the time. At the center, we have an example of the $k = 2$ case in which we would finalize in order to account for delays in attestation. The $k = 3$ example is mainly for the sake of illustration, as the network basically has to collude to orchestrate such contrived scenarios. In practice, Ethereum's specification for finalization in [11] does not even include finalization for $k \geq 3$.

With 1/2-synchrony and without implementation details that interfere with acceptance of messages, we should only get the case $k = 1$. We include the $k > 1$ cases to account for situations where network latency and attestation inclusion delays (see Section 8.4) are relevant. In fact, in the original Casper [7], finalization is only defined for the $k = 1$ case.[YZ]

If we define a set of blocks $F$ as finalized and prove safety for them, then safety remains true automatically if we change the definition of *finalized* to any subset of $F$, because safety is defined by the lack of incomparable pairs of finalized blocks. Thus, it does not hurt to define a very general class of blocks as finalized if we can prove safety. This is why we include all $k$ in our definition; the more general definition is easier to analyze.

## 4.5. Slashing Conditions

In this subsection, we add *slashing* conditions, analogous to those from Casper in Definition 3.1. We prove a couple of desired properties, after which we are ready to use these conditions to prove safety of Gasper in Section 5.

**Definition 4.5.** We define the following *slashing conditions*:

(S1) No validator makes two distinct attestations $\alpha_1, \alpha_2$ with $\mathsf{ep}(\alpha_1) = \mathsf{ep}(\alpha_2)$. Note that this condition is equivalent to $\mathsf{aep}(\mathsf{LE}(\alpha_1)) = \mathsf{aep}(\mathsf{LE}(\alpha_2))$.

(S2) No validator makes two distinct attestations $\alpha_1, \alpha_2$ with

$$\mathsf{aep}(\mathsf{LJ}(\alpha_1)) < \mathsf{aep}(\mathsf{LJ}(\alpha_2)) < \mathsf{aep}(\mathsf{LE}(\alpha_2)) < \mathsf{aep}(\mathsf{LE}(\alpha_1)).$$

**Remark.** An honest validator following the protocol will never "accidentally" violate the slashing condition (S1), because each validator is randomly assigned to exactly one committee in each epoch, and is thus asked to attest exactly once in that epoch. For (S2) the situation is more complicated. Under most normal circumstances, an honest validator will also never violate (S2), because of the following (heuristic) argument:

Usually (without high latency and not under attacks), $\mathsf{LJ}(\alpha)$ should either be equal to (when the view did not pick up enough attestations justifying a block in the current epoch) or exactly 1 less than (when the view has picked up enough attestations justifying a block in the current epoch) the last justified pair in the view of the validator when making the attestation $\alpha$. Thus, if an honest validator $V$ makes attestations $\alpha_1$ in time $t_2$ in epoch $j_2$ and $\alpha_2$ in time $t_1$ in epoch $j_2 > j_1$, because $\mathsf{view}(V, t_2) \supset \mathsf{view}(V, t_1)$, $\mathsf{LJ}(\alpha_2)$ (which is likely to just be the last justified pair in $\mathsf{view}(V, t_2)$, as just discussed) is likely to be later

than $\mathsf{LJ}(\alpha_1)$ as well, and the "nesting conditions" of (S2) would not be violated. However, in edge cases it is possible for $\mathsf{LJ}(\alpha)$ to have an earlier epoch than the last justified pair in the view of the validator when making the attestation $\alpha$, so the argument above is not a proof. Thus, in practice, the software for honest validators to use should have an extra layer of protection against breaking (S2), which is not hard to implement: one can simply make sure that the nesting condition is not violated before sending an attestation.

We now prove a very useful property of the protocol: unless we are in the unlikely situation that we have enough evidence to slash validators with at least $1/3$ of the total stake, in a view $G$ we may assume all elements in $J(G)$ have unique attestation epochs (in other words, the view sees at most one pair justified per epoch).

**Lemma 4.6.** In a view $G$, for every epoch $j$, there is at most 1 pair $(B, j)$ in $J(G)$, or there must exist 2 subsets $\mathcal{V}_1, \mathcal{V}_2$ of $\mathcal{V}$, each with total weight at least $2N/3$, such that their intersection violates slashing condition (S1).

*Proof.* Suppose we have 2 distinct pairs $(B, j)$ and $(B', j)$ in $J(G)$. That means in epoch $j$, more than a total stake of $2N/3$ attested with a checkpoint edge to $(B, j)$ and more than $2N/3$ stake attested with a checkpoint edge to $(B', j)$. These are our desired $\mathcal{V}_1$ and $\mathcal{V}_2$, as depicted in Figure 7. □

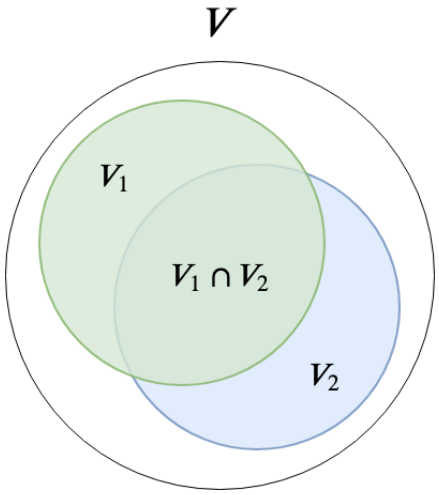



Figure 7: Two subsets of the validator set with at least $2N/3$ attestations of stake will overlap by at least $N/3$ stake.

## 4.6. Rewards and Penalties

A validator should be rewarded (i.e. have stake increased) for either including valid attestations in his/her proposed block (in the beacon chain specs [11], *proposer reward*) or attesting to the correct block which is justified and finalized as the chain grows (in [11], *attester reward*). Meanwhile, a validator should be penalized (have stake decreased) for violating slashing conditions.

The reward and penalty amounts can be adjusted based on the security level that needs to be achieved, and the game theory of the situation should be such that validators are incentivized to perform their tasks and to not violate slashing conditions. Additionally, it is worth considering more complex incentives, such as incentivizing validators to catch other misbehaving validators. While this makes for potentially interesting game theory work, adding a layer of analysis of these parameters is distracting for the scope of this paper, where we want to focus on the consensus aspect of Gasper. To be pragmatic, we abstract this analysis away by assuming that these reward and penalty mechanisms provide enough game-theoretical incentives such that:

- Honest validators follow the protocol.
- Any honest validator seeing a slashing condition violated will slash the dishonest validator.

# 5. Safety

## 5.1. Safety - Static Validator

Recall that our main goals are proving safety and liveness. In this section, we prove safety, which is done similarly as in Casper FFG [7]; the main differences are that we are using epoch boundary pairs (as opposed to Casper's checkpoint blocks) and our finalization definition is more complex. Thus, we need the following Lemma, which is necessarily more complicated than its counterpart idea in Casper.

**Lemma 5.1.** In a view $G$, if $(B_F, f) \in F(G)$ and $(B_J, j) \in J(G)$ with $j > f$, then $B_F$ must be an ancestor of $B_J$, or there must exist 2 subsets $\mathcal{V}_1, \mathcal{V}_2$ of $\mathcal{V}$, each with total stake at least $2N/3$, such that their intersection all violate slashing condition (S1) or all violate slashing condition (S2).

*Proof.* Anticipating contradiction, suppose there is a pair $(B_J, j)$ with $j > f$ and $B_J$ is not a descendant of $B_F$. By definition of finalization, in $G$, we must have $(B_F, f) \xrightarrow{\mathsf{J}} (B_k, f + k)$, where we have a sequence of adjacent epoch boundary pairs $(B_F, f), (B_1, f + 1), \ldots, (B_k, f + k)$.

Since $(B_J, j)$ is justified and $B_J$ is not a descendant of $B_F$, without loss of generality (by going backwards with supermajority links), we can assume $(B_J, j)$ is the earliest such violation, meaning that we can assume $(B_l, l) \xrightarrow{\mathsf{J}} (B, j)$ where $l < f$ but $j > f$ (here we are using Lemma 4.6, which tells us that no two justified blocks have the same $\mathsf{aep}()$, else we are done already with 2 validator subsets of weight $2N/3$ each violating (S1); this is why we do not worry about the equality case). Since $B_1, \ldots, B_k$ are all justified but are descendants of $B_F$, we know $B_J$ cannot be any of these blocks, so we must have $j > f + k$. This means that the view $G$ sees that some subset $\mathcal{V}_1$ of $\mathcal{V}$ with total stake more than $2N/3$ have made attestations justifying a checkpoint edge $(B_l, l) \to (B_J, j)$, so for any such attestation $\alpha_1$, $\mathsf{aep}(\mathsf{LJ}(\alpha_1)) = l$ and $\mathsf{ep}(\alpha_1) = j$. Similarly, $G$ also

sees that more than $2N/3$ weight worth of validators $\mathcal{V}_2$ have made attestations justifying $(B_F, f) \to (B_k, f+k)$ so for any such attestation $\alpha_2$, $\mathsf{aep}(\mathsf{LJ}(\alpha_2)) = f$ and $\mathsf{ep}(\alpha_2) = f + k$. Thus, for anyone in the intersection $\mathcal{V}_1 \cap \mathcal{V}_2$, they have made two distinct attestations $\alpha_1$ of the former type and $\alpha_2$ of the latter type. Because

$$l < f < f + k < j,$$

we know

$$\mathsf{aep}\left(\mathsf{LJ}\left(\alpha_1\right)\right) < \mathsf{aep}\left(\mathsf{LJ}\left(\alpha_2\right)\right) < \mathsf{ep}\left(\alpha_2\right) < \mathsf{ep}\left(\alpha_1\right),$$

which allows them to be provably slashed by (S2). □

We are now ready to prove safety.

**Theorem 5.2** (Safety)**.** In a view $G$, if $(B_1, f_1), (B_2, f_2) \in F(G)$ but $B_1$ and $B_2$ conflict, then $G$ has enough evidence to slash at least $N/3$ stake worth of validators.

*Proof.* This proof is similar to Theorem 1 in FFG [7]. Let $(B_1', j_1)$ and $(B_2', j_2)$ in $J(G)$, with supermajority links $(B_1, f_1) \overset{\mathsf{J}}{\to} (B_1', j_1)$ and $(B_2, f_2) \overset{\mathsf{J}}{\to} (B_2', j_2)$, which exist by definition of finalization. By Lemma 4.6, $j_1 \neq j_2$, or $G$ already has enough evidence to slash at least $N/3$ stake worth of validators.

Assuming $G$ does not have such evidence, then without loss of generality, $j_1 > j_2$, so $j_1 > f_2$ as well. By Lemma 5.1, $B_1'$ must be a descendant of $B_2$, which is impossible, as no two conflicting blocks can share a descendant. □

## 6. Plausible Liveness

This section is again similar to the treatment in Reference [7], Theorem 2. Because our setup is more complex, the results do not immediately follow from the Casper FFG paper, despite having the same main ideas.

**Theorem 6.1.** If at least $2N/3$ stake worth of the validators are honest, then it is always possible for a new block to be finalized with the honest validators continuing to follow the protocol.

*Proof.* Given any network view $G = \mathsf{view}(\mathsf{NW}, T)$ , consider the pair $(B, j) \in J(G)$ that appears with the highest $j$. If we are now in epoch $j' > j$, it is plausible that validators holding $2N/3$ of the stake justify the checkpoint edge $(B, j) \to (B', j')$. In particular, suppose our current slot is $i = 64j'$; then it is plausible (by following the protocol, so no slashing conditions are violated) for the block proposer to have the same view $G$, propose a child of $\mathsf{HLMD}(G)$, which is itself a descendent of $(B, j)$ by the design of the algorithm, to create a new block $B'$. Suppose no forks happen (more generally, it is plausible that the byzantine validators do not get any of the honest validators to attest to a conflicting block with $B'$). Then attestors attesting to $B'$ (or its descendents) will create a supermajority link $(B, j) \to (B', j')$, justifying $B'$.

By the same reasoning, in the next epoch, it is possible to create a supermajority link $(B', j') \to (B'', j' + 1)$. Now the finality conditions (with $k = 1$) are satisfied, so $B'$ becomes finalized, and we are done. □

The original Casper FFG is a "finality gadget" on top of a (presumed probabilistically live) blockchain, so its focus was not on the probabilistic liveness of the underlying chain, rather that the participants do not get into a situation where we are forced to slash honest validators to continue. Since our paper is also providing the underlying blockchain, we also want a probabilistic guarantee of liveness, which we will cover in Section 7. This is more technically difficult, so we will first prepare the work in Section B.

# 7. Probabilistic Liveness

In this section, we prove probabilistic liveness of our main protocol Gasper from Section 4, given some assumptions. Our proof consists of the following steps:

1. (assumptions lead to high weight after first slot) Under "good" conditions (such as having enough honest validators, synchrony conditions, etc. to be formalized) honest validators have a high probability of finding a block with a high weight after the first slot.

2. (high weight after first slot leads to high probability of justification) if we have a block with high weight after one slot, its weight "differential" over competitors is likely to increase throughout the epoch, meaning that this block (or a descendent) is likely to be justified.

3. (high probability of justification leads to high probability of finalization) if every epoch is likely to justify a block, then it is very likely for at least one block to be finalized in a stretch of $n$ epochs as $n$ increases.

This division is not just organizational; it also encapsulates the assumptions into different parts of the argument so that each part would remain true even if assumptions change. For example, the main theorem corresponding to the 3-rd item, Theorem 7.5, does not make any synchrony assumptions and only relies on a probability $p$ that a block gets justified (though synchrony assumptions may be required to bound $p$ given the previous parts). Also, we assume in the second part (as we do for simulations in Appendix B) that all validators have equal stake, whereas the third part does not.

This section is the most intricate and dependent on parameters: for one, the current planned implementation for Ethereum 2.0 actually avoids much of the analysis in this Section due to the attestation consideration delay (see Section 8.3); also, the bounds assume equal stake per validator, which may or may not be realistic depending on external factors. However, it is still worthwhile to include an analysis for the "pure" protocol because the strategy here (using concentration inequalities) is very generalizable to a whole class of these potential "slot"-based approaches to proof-of-stake, and we believe it captures the heart of what makes these approaches work probabilistically. In fact, "fixes" such as the aforementioned attestation delay exist primarily because of the types of attacks against the pure protocol that we mention, so it is useful for intuition-building to address these attacks even if the actual implementations avoid them.

Thus, the primary value of this section is as a proof-of-concept of the analysis involved in designing probabilistically-live protocols such as Gasper, as opposed to a mathematical guarantee of the actual implementation.

## 7.1. Probabilistic Liveness Preparation - The Equivocation Game

We define the *equivocation game* to be the following:

1. The game is parametrized by $(\mathcal{V}, a, \epsilon_1, \epsilon_2)$, where $\mathcal{V}$ is a set of validators and $(a, \epsilon_1, \epsilon_2)$ are real numbers that parameterize network / synchrony conditions (for ease of reading we push the details to the Appendix).

2. As in Gasper, $|\mathcal{V}| = N$ and each $V \in \mathcal{V}$ has some fixed stake $w(V)$. We assume the total stake is $N$ (so the average stake is 1) and the total amount of stake of honest validators is at least $2N/3$.

3. There are 2 options to vote for, which we call $O_1$ and $O_2$ for short. These are abstractions for voting on 2 conflicting blocks in Gasper, and the concept of "blocks" are not included in this equivocation game. We (meaning the honest validators) *win* if either $O_1$ or $O_2$ obtain at least $2N/3$ stake worth of votes, and we lose (i.e., we are in a state of equivocation) otherwise. Note that this is equivalent to the idea that there is a stake differential of $N/3$ between the two choices after the game.

The equivocation game is defined to capture the idea of Gasper with just 2 choices in one slot. Most of the necessary assumptions for liveness are encoded into the game so that the other liveness results in this section can be as independent as possible.

In the Appendix, we flesh out the details of the equivocation game and conduct the simulations of equivocation game with three regimes (a pessimistic regime, an optimistic regime, and a regime in between). The main idea is that under "reasonable" conditions $(\mathcal{V}, a, \epsilon_1, \epsilon_2)$, it is very likely that after the first slot, we win the game, meaning that we should expect one of the pairs in Gasper to have $2N/3$ worth of stake supporting it after the first slot.

**Remark.** Clearly, the concept of equivocation games can be generalized, and similar games could be used to study other proof-of-stake models. There are many directions (the protocols for honest validators, latency modeled by something more than the uniform distribution, etc.) which may be interesting to study for future work, but may be too distracting for the purpose of our paper.

## 7.2. High Weight after First Slot Leads to Justification

In this section, we focus on the $j$-th epoch $[T, T + C]$, where $T = jC$. Let $S = N/C$. We define the following set of assumptions, which we call $A(\epsilon)$:

- the network is $(1/2)-$synchronous starting at time $T = jC$;

- each validator has 1 stake;

- the total number of byzantine validators is equal to $N/3 - C\epsilon$, meaning the average number of byzantine validators in each slot's committee is $S/3 - \epsilon$.

For each $i \in \{0, 1, \ldots, C-1\}$, define $h_i$ to be the number of guaranteed honest attestors in slot $jC + i$ and $b_i$ to be the number of dishonest attestors in the slot. Let $S = N/C$, so for all $i$, $h_i + b_i = S$. We show that the distribution of honest / dishonest attestors should not stray too much from expectation in Proposition 7.2, using ubiquitous concentration inequalities:

**Proposition 7.1.** Let $\mathcal{X} = (x_1, ..., x_N)$ be a finite list of $N$ values with $a \leq x_i \leq b$ and let $X_1, ..., X_n$ be sampled without replacement from $\mathcal{X}$. The following hold:

$$\mathbb{P}\left(\sum_{i=1}^{n} X_i - n\mathbb{E}\left[X\right] \geq n\delta\right) \leq \exp\left(\frac{-2n\delta^2}{(1-(n-1)/N)(b-a)^2}\right)$$

$$\mathbb{P}\left(\sum_{i=1}^{n} X_i - n\mathbb{E}\left[X\right] \leq -n\delta\right) \leq \exp\left(\frac{-2n\delta^2}{(1-(n-1)/N)(b-a)^2}\right)$$

*Proof.* See e.g. [19]. □

**Proposition 7.2.** Suppose the assumptions $A(\epsilon)$ are met. Assume that $C = 2^L$. Then the conjunction of the following events (we purposefully skip $h_0$):

$$E_1 \ : \ h_1 \geq 2 \cdot \tfrac{S}{3}$$

$$E_2 \ : \ h_2 + h_3 \geq 2^2 \cdot \tfrac{S}{3}$$

$$E_3 \ : \ h_4 + h_5 + h_6 + h_7 \geq 2^3 \cdot \tfrac{S}{3}$$

$$\cdots \ : \ \cdots$$

$$E_L \ : \ h_{C/2} + \cdots + h_{C-1} \geq 2^L \cdot \tfrac{S}{3}$$

has probability

$$\mathbb{P}\left(\bigcap_{i=1}^{L} E_i\right) \geq 1 - \sum_{i=1}^{L} \exp\left(-\frac{2^i\epsilon^2}{\left(1 - \frac{2^{i-1}S-1}{2^L S}\right)S}\right) \geq 1 - \sum_{i=1}^{L} \exp\left(-\frac{2^i\epsilon^2}{S}\right).$$

*Proof.* We use the hypergeometric distribution model; that is, we consider the set $\mathcal{X} = (x_1, \cdots, x_N)$ representing the validators $\mathcal{V}$, where $x_j = 1$ if the $j$-th validator is honest and 0 otherwise. Let $X_1, \ldots, X_{2^{i-1}S}$ be $2^{i-1}S$ samples from $\mathcal{X}$ without replacement. Then for each $j$, $1 \leq j \leq 2^{i-1}S$, $\mathbb{E}\left[X_j\right] = 2/3 + \epsilon/S$, so:

$$\mathbb{E}\left[\sum_{j=1}^{2^{i-1}S} X_j\right] = 2^{i-1}S(2/3 + \epsilon/S) = 2^i S/3 + 2^{i-1}\epsilon.$$

We can then bound each $E_i$ (as a sum of $X_j$'s) by

$$\begin{aligned}\mathbb{P}\left(\sum_{j=1}^{2^{i-1}S} X_j \geq 2^i \frac{S}{3}\right) &= 1 - \mathbb{P}\left(\sum_{j=1}^{2^{i-1}S} X_j < 2^i \frac{S}{3}\right) \\ &= 1 - \mathbb{P}\left(\sum_{j=1}^{2^{i-1}S} X_j - \left(\frac{2^i S}{3} + 2^{i-1}\epsilon\right) < -(2^{i-1}S)\frac{\epsilon}{S}\right) \\ &\geq 1 - \exp\left(-\frac{2^i \epsilon^2}{\left(1 - \frac{2^{i-1}S-1}{2^L S}\right) S}\right),\end{aligned}$$

where the last inequality results from Proposition 7.1, recalling that $N = 2^L S$. The probability of each event $E_i$ when considered independently of each other is then bounded below by this value. Using the intersection bound on all $E_i$ we get the first inequality in the desired statement; the second inequality holds by comparing denominators. □

**Remark.** It suffices to also use e.g. Hoeffding's inequality with a binomial model where the validators are assigned as honest or byzantine with replacement. We leave it as an exercise to the reader that this method immediately gets the weaker bound

$$\mathbb{P}\left(\bigcap_{i=1}^{L} E_i\right) \geq 1 - \sum_{i=1}^{L} \exp\left(-\frac{2^i \epsilon^2}{S}\right).$$

While the binomial model is only an approximation, the model with replacement is strictly more mean-reverting than the model without, so the approximation is in the correct direction for us. This could be made rigorous with e.g. coupling methods. We use the weaker bound because it is algebraically cleaner and loses very little compared to the stronger bound.

It can be seen that the exponential terms in the result of proposition 7.2 is fairly small when $\epsilon$ is on the order of $\sqrt{S}$. For example, when $C = 64$ (a power of 2 actually makes Proposition 7.2 work cleanly, though it is certainly not a crucial part of the bound) and thus $S = 900$, picking $\epsilon = 30$ (meaning we have as many as 17280 byzantine validators out of 57600) gives a probability bound of around 85%; changing $\epsilon$ to 40 jumps the probability to around 97%.

**Lemma 7.3.** Suppose the assumptions in $A(\epsilon)$ are met. Then, if $\mathsf{view}(\mathsf{NW}, T + C)$ justifies a new block $B$ not in $J(\mathsf{view}(\mathsf{NW}, T))$, $B$ must be a descendant of the last justified block in $J(\mathsf{view}(\mathsf{NW}, T))$.

*Proof.* Recall that honest attestors wait for 1/2 time before attesting to their assigned slot. Thus, in the upcoming epoch $j$, all of the honest attestors in this epoch attest in the time period $[T+1/2, T+C]$. Because we have (1/2)-synchrony, it means their views during this period all include $\mathsf{view}(\mathsf{NW}, T)$. By the nature of Algorithm 4.1, this means all of their block proposals and attestations must be to either:

- a descendant of $B_J$, the last justified block in $\mathsf{view}(\mathsf{NW}, T)$, or
- a descendant of some new block justified in epoch $j$.

Our Lemma only fails in the second case, and if some new block in this epoch obtains 2/3 of the attestations of this epoch.

Luckily, the chicken-and-egg favors us. It is possible for byzantine validators to propose a new block $B'_J$ that's not a descendant of $B_J$. However, it would be impossible for $B'_J$ to receive enough votes in this epoch, as all of the attestations before $B'_J$ is justified must go to a descendant of $B_J$ by Algorithm 4.1. Thus, we know that if we justify a new block, it must be a descendant of $B_J$. □

**Theorem 7.4** (Justification Probabilistic Liveness)**.** Suppose the assumptions in $A(\epsilon)$ are met, and suppose we win the equivocation game corresponding to the first slot with probability $r$. Let $B_J$ be the last justified block in $J(\mathsf{view}(\mathsf{NW}, T))$. Then, $\mathsf{view}(\mathsf{NW}, T + C)$ will justify a new descendant of $B_J$ with probability at least

$$r - \sum_{i=1}^{L} \exp\left(-\frac{2^i \epsilon^2}{S}\right) - \frac{1}{3^{C-1}}.$$

*Proof.* We can think of the first slot of this epoch $[T, T + 1]$ as an equivocation game with $S$ validators ($h_0$ honest and $b_0$ byzantine) where all of the options are $B_J$ or a descendant[8] of $B_J$. Thus, with probability $r$, one of these options receives at least $2S/3$ attestations after slot $jC$. We call this winning block $B_w$ (ideally, $B_w$ is simply just the block $jC$, though this is not necessary). For sake of weighting in Algorithm 4.1, note that against any other option in the equivocation game, $B_w$'s weight is winning by at least $(2S/3 - S/3) = S/3$.

By the intersection bound, with probability at least

$$r - \sum_{i=1}^{L} \exp\left(-\frac{2^i \epsilon^2}{S}\right)$$

we also satisfy the events in Proposition 7.2. We now show that when these events are satisfied, **all remaining honest validators in slots** $(jC + 1), \ldots, (jC + C - 1)$ **will vote for** $B_w$ **or a descendant.**

Consider slot $jC + 1$. Because of $E_1$, we know $b_1 < S/3$, so even if all $b_1$ potentially byzantine actors conspire to vote for some other option, the weight advantage of $B_w$ cannot be diminished to 0 by the time the honest validators vote at time $(jC + 1.5)$. This means that all honest validators will keep attesting to $B_w$ or a descendant block (an honest proposer in slot $jC + 1$ would propose block $jC + 1$ as a descendant of $B_w$, for example). By $E_1$, we know that $B_w$ gains at least $2S/3$ weight while a rival block gains at most $S/3$ weight during this slot, which means that the weight differential preferring $B_w$ changes by at least $(2S/3 - S/3) = S/3$. This means by the end of slot $jC + 1$, $B_w$ is now winning with at least weight $S/3 + S/3 = 2S/3$.

[8] As stated in the proof of Lemma 7.3, it is possible that there are options that are not, but they will never receive attestations from honest validators, so it would be strictly worse for dishonest validators to create such options.

Now consider $E_2$ and the next 2 slots, $jC+2$ and $jC+3$. Between them, we know $h_2 + h_3 < 2S/3$, so even if all the byzantine actors conspire, they cannot destroy the winning differential of $B_w$, which by the end of these 2 next slots will be winning with at least weight $(2S/3 - 2S/3 + 4S/3) = 4S/3$.

Inductively, this logic continues for all remaining slots in the epoch (specifically, all honest validators attest to $B_w$ or a descendant. Here the structure of Algorithm 4.1 is important because while honest attestors may (and probably will) attest to new blocks, the weight of their attestations are added to that of $B_w$ as well.

We have thus concluded that during all remaining slots, $B_w$ accumulates all attestations from honest validators. After slot $jC$, it has at least $2S/3$ votes with probability $r$. Then, we know it picks up weight at least $h_1 + h_2 + \cdots + h_{C-1}$, for a total weight of

$$\frac{2}{3}S + \sum_{i=1}^{L} \frac{2^i}{3}S = \frac{2}{3}S + \frac{2}{3}\left(2^L - 1\right)S = \frac{2^{L+1}}{3}S = \frac{2}{3}N,$$

so we indeed achieve enough weight for a supermajority link.

To prove $B_J$ is different from $B_w$, we need at least one honest validator to propose a new block on top of $B_J$ before or in the first slot of epoch $\mathsf{aep}(B_w)$. Since $B_J$ receives more than $\frac{2}{3}$ of votes in epoch $\mathsf{aep}(B_j)$, there are more than $\frac{2}{3}$ honest validators in $\mathsf{aep}(B_j)$. The chance of **no** honest validator proposing a block on top of $B_J$ in $\mathsf{aep}(B_j)$ is then bounded above by $(\frac{1}{3})^{C-1}$, which is vanishingly small. □

## 7.3. Probabilistic Justification Leads to Probabilistic Finalization

The main theorem is the following:

**Theorem 7.5** (Finalization Probabilistic Liveness)**.** Assume the probability of justifying a block (as in Proposition 7.2) is independently $p \geq 1/2$ for each epoch, the probability of failing to finalize a block in the next $n$ epochs approaches 0 exponentially as a function of $n$.

*Proof.* Recall from the definition of finalization that finalization happens if a chain of consecutive epoch boundary blocks are justified; in particular, if one justified block finalizes an adjacent epoch boundary block, they are finalized (with $k = 1$ in the definition of finalization). For this proof, let us call this $k = 1$ *finalization*.[YZ]

We consider an epoch a "success" if a block is justified in the epoch, and a "failure" otherwise. Thus, if 2 adjacent epochs are "successful," we finalize a block. Therefore, the probability of not getting a ($k = 1$) finalization in $n$ epochs is the probability that no 2 adjacent epochs out of the next $n$ are successful. Using the independence assumption, this has the probability

$$\sum_{i \geq \frac{n}{2}}^{n} \binom{i+1}{n-i} (1-p)^i p^{n-i},$$

because for a particular $i \geq n/2$, there are $\binom{i+1}{n-i}$ ways to select $i$ failures.

Since $p \geq 0.5$, we know $p \geq (1-p)$, so the sum is bounded above by

$$\left( \sum_{i=n}^{\lfloor \frac{n}{2} \rfloor} \binom{i+1}{n-i} \right) (1-p)^{n/2} p^{n/2}$$

It is well-known (by e.g. induction) that:

$$\sum_{i=n}^{\lfloor \frac{n}{2} \rfloor} \binom{i+1}{n-i} = \binom{n+1}{0} + \binom{n}{1} + \binom{n-1}{2} + \cdots = F_n,$$

the $n$-th Fibonacci number, which we know is of the form

$$F_n = \frac{1}{\sqrt{5}} \left[ \frac{(1+\sqrt{5})}{2} \right]^n - \frac{1}{\sqrt{5}} \left[ \frac{(1-\sqrt{5})}{2} \right]^n$$

The second term vanishes as $n \to \infty$, so our desired quantity is bounded above by (times a constant)

$$\left[ \frac{(1+\sqrt{5}) \sqrt{p(1-p)}}{2} \right]^n$$

We have the bound $\sqrt{p(1-p)} \leq 1/2$ (by, e.g. AM-GM inequality), so the failure rate is bounded above by

$$\frac{1}{\sqrt{5}} \left( \frac{1+\sqrt{5}}{4} \right)^n,$$

which goes to 0 as $n \to \infty$. Finally, recall that we are only looking at $k = 1$ finalization, so the chances of $k = 2$ finalization can only theoretically increase (though in practice they should not happen in good synchrony conditions anyway).[YZ]

□

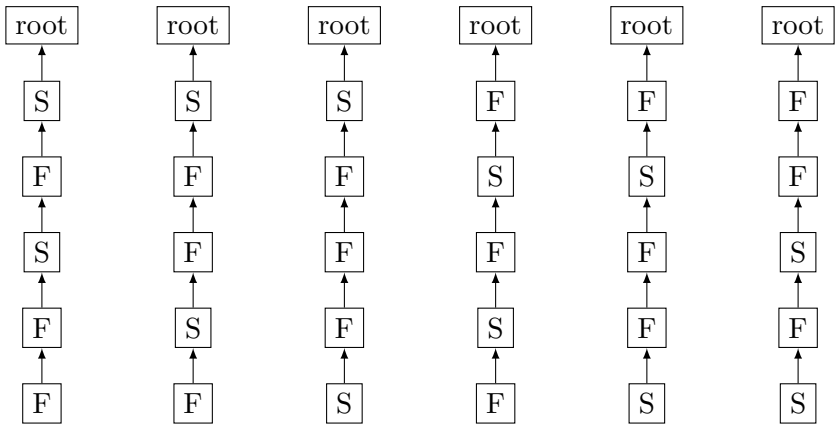



Figure 8: The cases where we **fail** to finalize a block in $n = 5$ epochs with 3 failing epochs. "S" denotes success and "F" denotes failsure (with respect to Proposition 7.2.

Theorem 7.5 is "agnostic" of the justification probability $p$ and how we obtained it. This means if we tweak the protocol, change the assumptions, etc. and obtain different bounds/estimates for $p$ than we do from the current Theorem 7.4, our result still holds. For this reason, we use $p$ as an **input** to Theorem 7.5 instead of reusing our actual bound in Theorem 7.4. A secondary effect of abstracting away justification liveness is that **the technique here can be used to prove finalization probabilistic liveness for general 2-stage protocols, not just** Gasper (it could even be extended easily to protocols with 3- or more stages, though the bounding would be different). Regardless of the protocol (or $p$, as long as it is at least 0.5), as $n$ increases, the probability finalizing a block increases rapidly, getting around 99% even for $p = 0.5$ after $n = 20$ epochs. See Table 1 for some computations.

Table 1: The probabilities of **not** finalizing blocks in $n$ epochs, with $p$ the probability of justification within each epoch.

| n | p | probability of not finalizing any block in n epochs |
|---|---|---|
| 2 | 0.5 | 0.75 |
| 5 | 0.5 | 0.40625 |
| 7 | 0.5 | 0.265625 |
| 10 | 0.5 | 0.140625 |
| 20 | 0.5 | 0.016890525817871094 |
| 2 | 0.66 | 0.5644 |
| 5 | 0.66 | 0.18460210239999997 |
| 7 | 0.66 | 0.08322669164799996 |
| 10 | 0.66 | 0.025351233503186934 |
| 20 | 0.66 | 0.0004854107646743359 |

**Remark** (Relationship with Plausible Liveness)**.** Careful readers will see that

we technically get plausible liveness "for free" from probabilistic liveness. So why do we treat them separately in this paper? For one, plausible liveness is an immediate consequence from the rules of the protocol Gasper and requires very few assumptions, while probabilistic liveness is dependent on probabilistic assumptions (such as network synchrony) and is more fragile with slight changes to the protocol. In particular, one part of our treatment assumes that all validators have the same stake, an assumption that can probably be relaxed but would require much more work. Thus, we choose to present plausible liveness separately to emphasize that even if these assumptions are not satisfied "in real life," plausible liveness remains. Secondly, the emphasis of plausible liveness is that "honest validators will never **be required by the protocol** to voluntarily slash themselves to continue" and the emphasis of probabilistic liveness is "new blocks will probably be justified/finalized quickly;" these are different takeaways.

# 8. Practice vs Theory

Consider reorganizing this section?[YW]

Ethereum's practical implementation in [11] has a few design decisions that are different from Gasper and is driven by some desirable features of the beacon chain. Our protocol Gasper is meant to be a "clean" protocol that captures the theoretical core of the beacon chain, which is more mathematically tractable to study. In this section, we consider the differences between Gasper and the actual implementation. We seek to pragmatically cover the main concepts without getting lost in the much messier analyses a rigorous study of adding all of these details would require.

## 8.1. Sharding

This paper is motivated by the Ethereum 2.0 *beacon chain*, which is the "main" blockchain in the Ethereum 2.0 design that stores and manages the registry of validators. In this implementation, a *validator* is a registered participant in the beacon chain. Individuals can become a validator of the beacon chain by sending Ether into the Ethereum 1.0 deposit contract. As in Gasper, validators create and attest to blocks in the beacon chain. Attestations are simultaneously proof-of-stake votes for a beacon block (as in our design) but also availability votes for a "shard block," which contains data in a different "shard chain." This concept of *sharding* creates interesting engineering and mathematical questions outside the scope of our paper, which is limited to the beacon chain.

## 8.2. Implementing the View

In a "pure" protocol such as Gasper, we can treat views and related concepts as abstract mathematical concepts and validators as perfectly reasoning agents with infinite computational power. In practice, validators will not be directly reasoning with a graphical data structure of a view; instead, they will use software to parse the view given to them and follow the protocol. Thus, in the actual

implementation [11], validators run a program that updates the "store", which is basically a representation of the view. The store, as the input for LMD GHOST fork choice rule, is updated whenever a block or an attestation is received. The beacon chain also keeps track of a "state," a derived data structure from the view that keeps track of stake-related data.

Obviously, mistakes when interpreting these structures and their updates may cause issues with safety and liveness not related to those coming from Gasper itself. Even though in our work we limit our analysis to the mathematical parts of the protocol, we remind the reader that these other issues are also important; security holes arising from from a carelessly implemented protocol at the software level are not protected by the mathematical guarantees of Gasper.

## 8.3. Attestation Consideration Delay

In the Ethereum 2.0 beacon chain implementation, when a validator is supposed to attest at slot $N$, he/she runs the LMD GHOST fork-choice rule, but only considers attestations that are at least 1 slot old. In our terms, the view used as input to the fork-choice rule contains all valid blocks up to slot $N$ but only contains valid attestations up through slot $N-1$; whereas in Gasper we consider all blocks and attestations in the validator's view.

This one-slot delay protects validators from a certain class of timing attacks in which byzantine validators eagerly broadcast slot $N$ attestations rather than waiting for the $N + 1/2$ time when they are "supposed" to attest. This allows them to theoretically take advantage of the network latency to split the vote of the honest validators to keep the chain in a state of equivocation. This is an important attack to counter as publishing attestations at the "wrong" time in a block is not a slashable offense (since the adversary can, a priori, fake timestamps; though it is possible to design a sophisticated blockchain based on Gasper that makes faking timestamps harder).

Luckily, we covered the one-shot delay attack in our analysis of the *equivocation game* (see Appendix B), and conclude it is plausible for the attack to be effective in the pessimistic regimes. However, the probability byzantine validators succeed in this particular type of attack is greatly lowered even in pessimistic regimes if we consider the attestation consideration delay.

Specifically, this delay makes the equivocation game winning rate as analyzed in Theorem 7.4 at least $r = 2/3$. As long as an honest validator creates a block (with probability at least $2/3$ in the worst case), then all honest validators will vote for that block and create a winning weight of at least $2N/3 - N/3 = N/3$ for that block. This argument heuristically shows that the probabilistic liveness of the actual protocol comes with even better guarantees than that of Gasper. A rigorous analysis / proof of this intuition may be interesting future work.

## 8.4. Attestation Inclusion Delay

In Gasper, when a validator proposes a block, he/she includes all attestations in his/her view. In the actual Ethereum 2.0 beacon chain, there is an integer

parameter for the *attestation inclusion delay* (say $n$) such that when a validator proposes a block, he/she only includes attestations that were made $n$ slots ago.

The purpose of the attestation inclusion delay is to prevent centralization and reward advantage in the case where block time is super short. Currently, the beacon chain expects a new block every slot (6 seconds). If the slot time is short and if the latency of attestation propagation is high for normal nodes in the network, then highly-connected nodes might be able to get attestation data faster and be able to publish them faster, which may correspond to various advantages in real life. For example, this can happen when the slot time is less than twice as long as the average attestation propagation time and if the latency is much higher than half slot time.

The enforced delay allows attestations to disseminate more widely before they can be included in blocks. Thus, nodes have more of an equal opportunity to capture the attestation inclusion rewards for proposing blocks. The plan is to tune the attestation inclusion delay (in the range of 1 to 4 slots) depending on real-world network data. Smaller values improve the transaction processing speed, and larger values improve decentralization. Note that increasing the slot time is an alternative method to promote decentralization.

## 8.5. A Four-Case Finalization Rule

While Definition 4.4 captures the "clean" version of the mathematical idea of finalization, Ethereum 2.0 uses a "reduced" version only looking at the last 4 epochs for practicality. In particular, let $B_1, B_2, B_3, B_4$ be epoch boundary blocks for consecutive epochs, with $B_4$ being the most recent epoch boundary block.

1. If $B_1, B_2$ and $B_3$ are justified and the attestations $\alpha$ that justified $B_3$ have $LJ(\alpha) = B_1$, we finalize $B_1$.
2. If $B_2, B_3$ are justified and the attestations $\alpha$ that justified $B_3$ have $LJ(\alpha) = B_2$, we finalize $B_2$.
3. If $B_2, B_3$ and $B_4$ are justified and the attestations $\alpha$ that justified $B_4$ have $LJ(\alpha) = B_2$, we finalize $B_2$.
4. If $B_3, B_4$ are justified and the attestations $\alpha$ that justified $B_4$ have $LJ(\alpha) = B_3$, we finalize $B_3$.

These are special cases of Definition 4.4 for cases $k = 1$ (the first two) and $k = 2$. The idea here is that Ethereum 2.0 will only honor attestations for up to 2 epochs. This means that only epoch boundary blocks up to 2 epochs in the past can be newly justified (and thus finalized), which gives the 4 cases. See Figure 9.

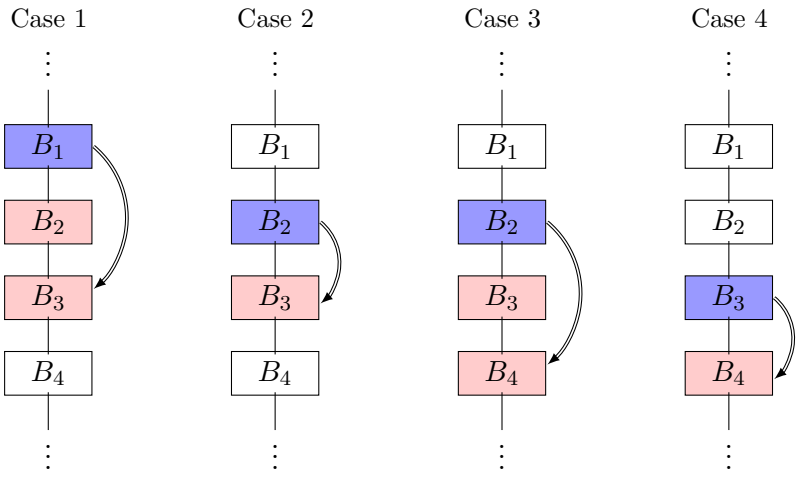



Figure 9: From left to right, example of cases 1 to 4 respectively. Both red and blue colors indicate justified blocks, where the blue blocks are blocks that are now finalized because of the observed justification, shown by double arrows.

## 8.6. Safety - Dynamic Validator Sets

In our previous discussion about safety in Section 5, we assumed *static* validator sets, meaning that the set of validators cannot change over time. Recall that our main result, Theorem 5.2, tells us we are able to catch $N/3$ weight worth of validators violating the slashing conditions if safety is broken. However, in practice, we would like to support *dynamic* validator sets, meaning that validators are allowed to *activate* (enter) and *exit* the validator set $V$ over time. This means byzantine activators can act maliciously, but then leave to avoid their stake being slashed. This setup reduces the number of "active" validators we can punish for violating slashing conditions.

**Definition 8.1.** We define "activated" and "exited" validators with respect to two validator sets, $\mathcal{V}_1$ and $\mathcal{V}_2$, assuming that $\mathcal{V}_2$ is a validator set later in time compared to $\mathcal{V}_1$. We define $A(\mathcal{V}_1, \mathcal{V}_2)$, the validators who *activated (from $\mathcal{V}_1$ to $\mathcal{V}_2$)*, to be the set of validators who are not in $\mathcal{V}_1$ but are in $\mathcal{V}_2$, and $E(\mathcal{V}_1, \mathcal{V}_2)$, the validators who *exited (from $\mathcal{V}_1$ to $\mathcal{V}_2$)*, to be the set of validators who are in $\mathcal{V}_1$ but not in $\mathcal{V}_2$.

First, we note that our key ideas from Section 5 still hold. To make it clear in our language, we rewrite them as follows:

**Lemma 8.2.** Suppose we allow dynamic validator sets. In a view $G$, if $(B_1, f_1)$ and $(B_2, f_2)$ in $F(G)$ conflict, then there must exist 2 justified pairs $(B_L, j_L)$ and $(B_R, j_R)$ in $G$ and 2 subsets $\mathcal{V}_1 \subset \mathcal{V}(B_L), \mathcal{V}_2 \subset \mathcal{V}(B_R)$, each with weight at least $2/3$ of the stake of its corresponding attestation epoch, such that their intersection $\mathcal{V}_1 \cap \mathcal{V}_2$ violates (S1) or (S2).

*Proof.* The proof is essentially identical to that of Lemma 5.1 and the reduction of Theorem 5.2 to it. The only difference is that we must rephrase the conditions of Lemma 5.1 as conditions about the validator sets at the time of their attestations.

As in the proof of Theorem 5.2, if we have 2 conflicting blocks $(B_1, f_1)$ and $(B_2, f_2)$, then either $f_1 = f_2$ or $f_1 \neq f_2$. If the former is true, then we can set $(B_L, j_L) = (B_1, f_1)$ and $(B_R, J_R) = (B_2, f_2)$ to satisfy our claim. If $f_1 \neq f_2$, without loss of generality, $f_1 < f_2$. We know that since $(B_2, f_2)$ is finalized, $(B_2, f_2) \xrightarrow{\mathsf{J}} (B_2', j_2) \in J(G)$, and we observe that $B_1$ is not an ancestor of $B_2'$. We now have the setup for Lemma 5.1, whose proof concluded there must be some pair of pairs (not necessarily $(B_F, f)$ or $(B_J, j)$ themselves), each of whose attestations have $2N/3$ stake, and whose intersections violate (S1) or (S2). □

In our setup, we suppose that the last finalized block observed in a view was $B_0$ at time $T_0$, and the view contains two conflicting finalized blocks $B_L$ and $B_R$ that were published at time $T_L$ and $T_R$ respectively. Let the blocks $B_0, B_L, B_R$ have validator sets $\mathcal{V}_0, \mathcal{V}_L, \mathcal{V}_R$ respectively. Let $a_L = w(A(\mathcal{V}_0, \mathcal{V}_L))$ and $e_L = w(E(\mathcal{V}_0, \mathcal{V}_L))$ be the total weight of activations and exits between $\mathcal{V}_0$ and $\mathcal{V}_L$ respectively, and similarly define $a_R$ and $e_R$. Our goal is to see what stake worth of validators can be provably slashed, in terms of $a_L$, $a_R$, $e_L$, *and* $e_R$, which are numbers we can control via our activation and exit policies.

**Theorem 8.3.** If a view contains two conflicting finalized blocks $B_L$ and $B_R$ as defined in the setup above, then the view has enough evidence to slash validators with total weight at least

$$X - w(\mathcal{V}_L)/3 - w(\mathcal{V}_R)/3,$$

where

$$X = \max(w(\mathcal{V}_L) - a_L - e_R, w(\mathcal{V}_R) - a_R - e_L).$$

*Proof.* By assumption, $B_L$ and $B_R$ are finalized, so by Lemma 8.2 we have quorums $Q_L \subset \mathcal{V}_L$ and $Q_R \subset \mathcal{V}_R$, both with at least 2/3 of their corresponding attestations, such that their intersection can be slashed.

We denote the different sets created by the overlap of our 3 validator sets to be $A, B, C, D, E, F, G$, as in Figure 10.

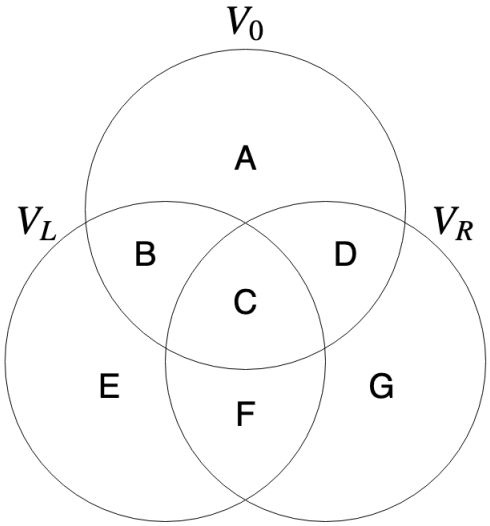



Figure 10: A Venn diagram of the initial validator set $\mathcal{V}_0$ and two validator sets $\mathcal{V}_L$ and $\mathcal{V}_R$, the letters A through G pictorially represent how many validators are in each overlap.

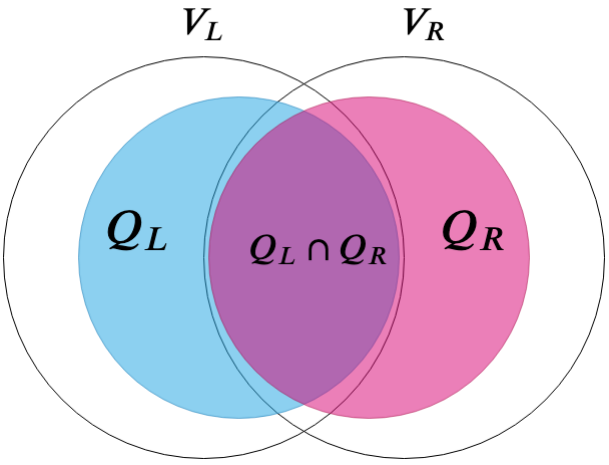



Figure 11: The quorums $Q_L$ and $Q_R$ have overlapping validators between the conflicting blocks $B_L$ and $B_R$.

Next, we wish to bound the intersection of the quorums $Q_L$ and $Q_R$ as depicted in Figure 11. Let $\mathcal{V}_{LR} = \mathcal{V}_L \cap \mathcal{V}_R = C \cup F$, we have:

$$\begin{aligned} w(Q_L \cap Q_R) &\geq w(Q_L \cap \mathcal{V}_{LR}) + w(Q_R \cap \mathcal{V}_{LR}) - w(C \cup F) \\ &\geq 2w(\mathcal{V}_L)/3 - w(B \cup E) + 2w(\mathcal{V}_R)/3 - w(D \cup G \cup C \cup F) \\ &= 2w(\mathcal{V}_L)/3 + 2w(\mathcal{V}_R)/3 - (w(B \cup C \cup D \cup E \cup F \cup G)) \\ &= w(C) + w(F) - w(\mathcal{V}_L)/3 - w(\mathcal{V}_R)/3. \end{aligned}$$

Let $X = w(C) + w(F)$. We know

$$\begin{aligned} X &= w(\mathcal{V}_L) - w(B \cup E) \\ &\geq w(\mathcal{V}_L) - w(A \cup B \cup E \cup F) \\ &= w(\mathcal{V}_L) - a_L - e_R, \end{aligned}$$

and similarly

$$X \geq w(\mathcal{V}_R) - a_R - e_L. \qquad \square$$

If we wanted to avoid negative signs in front of the validator sets, we could e.g. use a linear combination of the two bounds for $X$ to get the bound

$$w(\mathcal{V}_L)/3 - (2a_L/3 + 2e_R/3 + a_R/3 + e_L/3)\,.$$

As a sanity check, note that in the extreme case (no exits or activations), all the $a$'s and $e$'s are zero, so we recover the original bound of $w(\mathcal{V}_L)/3 = N/3$.

The power of the bound depends on our policies on activating and exiting. Here are some examples:

- If we allow a constant stake of $k$ worth of new validators to activate per epoch, then we can bound

  $$a_L, a_R \leq k\,(T_{now} - T_0)\,,$$

  where $T_{now}$ is the epoch when we observe the conflicting blocks in our view and $T_0$ is the epoch of the last finalized block. We can do the same with exiting and obtain bounds on $e_L$ and $e_R$, possibly with a different constant in the same prototocol.

- If we allow activating proportional to validator stake (so e.g. in one epoch, up to $k \cdot w(\mathcal{V})$ can activate) then the bounds become exponential:

  $$a_L, a_R \leq w(\mathcal{V})(1+k)^{T_{now}-T_0};$$

  In the limit this becomes an exponential bound.

Our bounds can also take care of latency; if we are prepared for up to $\delta$ epochs of latency, then we effectively add $\delta$ to computations where $(T_{now} - T_0)$ occur in our bounds of $a$'s and $e$'s, which we can then feed back into Theorem 8.3.

These bounds show **a tradeoff between the flexibility of entering and exiting the blockchain and the ability to catch malicious actors.** Our static result, Theorem 5.2, has the intuition "if our protocol breaks safety, we can provably slash 1/3 worth of validators" to imply safeness of our protocol as long as we have a strong enough belief in the validators. Our dynamic result, Theorem 8.3, is of the more nuanced form "if our protocol breaks safety, then as long as the attackers cannot create a fork with long branch lengths, we can provably slash a little less than 1/3 worth of validators." This means pure Gasper is susceptible to attacks such as malicious actors shutting down the whole

network for many[9] epochs and then suddenly appearing with a conflicting fork, such that we cannot slash much stake worth of validators since they have already exited during this time. Such situations can be safeguarded in practice by simply not including old attestations into one's view; the current Ethereum 2.0 spec, for example, does not accept attestations 2 or more epochs old, which avoids this attack.

### 8.7. Extreme Cases; Hard Forks

Even though we have fairly unconditional plausible liveness with a $\frac{2}{3}$ majority of honest stakeholders and probabilistic liveness under "good" conditions, it is important to take into account of worst-case scenarios. In the case of extended forking and lack of finality due to lots of *non-live* (non-participating) validators who are not necessarily malicious (and thus not slashed), the Ethereum 2.0 beacon chain has a mechanism by which *live* (participating) validators on a fork retain their stake, whereas the non-live validators "bleed" stake such that the live validators eventually become a $\frac{2}{3}$ majority, in time on the order of magnitude of weeks. Such a case may happen if e.g. the global internet is partitioned such that 50% of validators are on one side and 50% on the other.

A different extreme case is where a chain "flip-flops" continuously such that there is no distinct partition, and instead the majority just keeps switching forks each epoch. The ultimate backup plan for this situation or more severe cases is making *hard forks* (or *manual forks*), points at which the community running a blockchain chooses to alter the consensus rules (famous such cases include Bitcoin vs. Bitcoin Cash, or ETH vs. ETH Classic). This might be due to upgrading or adding features, fixing critical issues in production, or addressing a fractured community or political split. Usually, this results in a new set of protocol rules being run starting at some particular point in the blockchain. It is an important engineering problem, again outside of our mathematical scope, that the state transition and fork-choice functions can handle these changes smoothly. In most considerations the HLMD GHOST fork-choice rule would be untouched. However, when the alterations are "deep" inside the logic, considerations beyond the built-in forking mechanism would have to be taken and manual forks might be implemented.

## 9. Conclusion

We presented an abstract protocol, Gasper, that combines LMD GHOST and Casper FFG for a full proof-of-stake based blockchain design. Our goal was to separate the "mathematically clean" part of the Ethereum 2.0 design from the implementation details, which we discussed separately in Section 8. This work also serves as a "proof-of-concept" of the Casper finality gadget applied to a complete blockchain protocol. Finally, the techniques we used in e.g. proving

[9] For reference, it takes roughly 2.5 months in the concrete protocol to turn over 1/3 of the validator set, given the queuing mechanism in the Ethereum implementation, which is on the order of 20000 epochs

probabilistic liveness and/or forming the equivocation game may be useful for other such protocols, even though the actual Ethereum 2.0 blockchain design skirts much of the worst-case analysis there with "patches" such as found in Section 8.

There exist many other proof-of-stake based protocols in the cryptocurrency space. Some examples include:

- Tendermint [4]: a very "pure" design that is a simplification of PBFT applied to a blockchain proof-of-stake context. This design favors safety over liveness. All validators participate in every consensus round and no consensus happens faster than those rounds, with progress only possible if $\geq 2/3$ stake worth of validators are online.

- Casper CBC [5]: an alternative proposal to Casper FFG, focusing on mathematical correctness by construction. Casper CBC does not have fixed in-protocol thresholds, based on an emergent approach to safety arising from nodes following the majority of what other nodes have done in the past and making finality inferences about blocks. Besides being a proposed protocol, Casper CBC is also meant to be a general framework for analyzing consensus designs.

- Hotstuff (v6) [21]: has many similar properties as Casper FFG, with a flexible mathematical framework; one key difference is that Hotstuff uses an exponential backoff mechanism to be able to make progress under arbitrary network delays.

- Ouroboros [15]: a protocol under the "synchronous" school which assumes synchrony for fault-tolerance (and thus gets stronger 50% bounds for fault-tolerance). It uses the longest-chain rule and reward mechanisms to incentivize participation and prevent passive attacks. In Ouroboros Praos [15], the protocol is updated in response to vulnerabilities against "message delay" attacks. In Ouroboros Genesis in [1], the protocol is further expanded to allow for the ability for a participant to bootstrap the consensus from the genesis block (hence the name), proving globally universally composable (GUC) security against a 50% adversary.

- Snow White [10]: a protocol where the committee leader can extend the blockchain with a block that includes a reference to the previous block, similar to block dependencies, and a nonce that can be used to verify the block's validity against some a-priori difficulty constant. Valid timestamps must increase and "any timestamp in the future will cause a chain to be rejected". Participants only accept incoming chains that have not been modified too far in the past, similar to accepting views within a certain number of epochs.

- Nxt [9]: a proof-of-stake protocol that has a finite number of tokens. Nxt uses a stake-based probability for the right to generate a block, but it imposes transaction fees to circulate currency since it does not generate new

tokens. Nxt contrasts themselves from Peercoin, noting that Peercoin's algorithm grants more power to miners who have had coins on the network for a long time. The protocol also describes certain restrictions on block creation and transferring tokens to prevent standard attacks and moving stake between accounts.

- Thunderella [18]: a proof-of-stake protocol that aims to achieve optimal transaction verification relative to message delay, assuming "good conditions" that the leader of a committee is honest and has a super-majority of honest validators. A new leader will be implemented to build upon a back-up network in the event of a stall. The protocol also allows for dynamic validators with cool-down periods.

- Dfinity [13]: beacon-notarization protocol that works with both proof-of-work and proof-of-stake. The system uses a random beacon that selects block proposers, and a decentralized notary chooses the highest-ranked block based on a criteria described in [13]. Their analogues of *validators* and *committees* are called *replicas* and *groups* respectively, and the decentralized notary depends on the same Byzantine fault-tolerance to notarize blocks and reach consensus. They only have a passive notion of finalization, but the notarization process is quite fast, and blocks do get published in a timely manner. Replicas do need permission to leave the chain.

Our goal is not to show that our design is strictly better than any of these other designs. All of these protocols contain tradeoffs based on different design goals and assumptions about the network. Our design is guided by a balance between simplicity, understandability, and practicality, placing a mixed emphasis on safety and liveness. For example, if the context of a proposed blockchain is one where speed is less important than safety, then one may wish to e.g. select or modify a proposal more in the direction of Tendermint.

## Acknowledgments

This project is made possible by the SJSU Math and Statistics Department's CAMCOS (Center for Applied Mathematics, Computation, and Statistics) program (author Yan Zhang is also the acting director of CAMCOS). We thank the Ethereum Foundation and the San Jose State University Research Foundation for support for this program. We also thank Carl Beekhuizen, Dankrad Feist, Brian Gu, Brice Huang, Juan Sanchez, Yi Sun, Mayank Varia, and Sebastien Zany for helpful comments.

## References

[1] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930. ACM, 2018.

[2] L. Baird, M. Harmon, and P. Madsen. Hedera: A public hashgraph network & governing council. 2018. https://www.hedera.com/hh-whitepaper-v1.4-181017.pdf.

[3] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the Association for Computing Machinery (JACM)*, 32(4):824–840, 1985.

[4] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[5] V. Buterin. A cbc casper tutorial. 2018. https://vitalik.ca/general/2018/12/05/cbc_casper.html.

[6] V. Buterin. Sharding. https://vitalik.ca/files/Ithaca201807_Sharding.pdf, 2018.

[7] V. Buterin and V. Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[8] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *Operating Systems Design and Implementation*, volume 99, pages 173–186, 1999.

[9] N. Community. Nxt whitepaper. 2018. https://nxtwiki.org/wiki/Whitepaper:Nxt.

[10] P. Daian, R. Pass, and E. Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. *Cryptology ePrint Archive*, 2017.

[11] E. Developers. Ethereum 2.0 phase 0: The beacon chain. 2019. https://github.com/ethereum/eth2.0-specs/blob/dev/specs/core/0_beacon-chain.md.

[12] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the Association for Computing Machinery (JACM)*, 35(2):288–323, 1988.

[13] T. Hanke, M. Movahedi, and D. Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[14] A. Jain, S. Arora, Y. Shukla, T. Patil, and S. Sawant-Patil. Proof of stake with casper the friendly finality gadget protocol for fair validation consensus in ethereum. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 3(3):291–298, 2018.

[15] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[16] S. Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[17] K. Palmskog, M. Gligoric, L. Peña, B. Moore, and G. Rosu. Verification of casper in the coq proof assistant. Technical report, 2018.

[18] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[19] R. J. Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, 2(1):39–48, 01 1974.

[20] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[21] M. Yin, D. Malkhi, G. G. G. Reiter, Michael K., and I. Abraham. Hotstuff: Bft consensus in the lens of blockchain. *CoRR*, abs/1803.05069, 2018.

# A. Technicalities of Views

In Section 2, we defined *view* in a streamlined treatment. We provide a more detailed treatment in this section that offers further intuition.

First, we give a different definition from *view* that should seem more intuitive to some readers. We define the *full view* of a validator $V$ at a given time $T$ to be the set of all messages (and timestamps) that have been seen by $V$ by time $T$. We can denote the full view as $\mathsf{fv}(V, T)$. Similarly to the network view $\mathsf{view}(\mathsf{NW}, T)$, we also define a "God's-eye-view" $\mathsf{fv}(\mathsf{NW}, T)$, the *full network view*, as the collection of all messages any validator has broadcast at any time to the network. As with views, for any validator $V$ and any given time $T$, $\mathsf{fv}(\mathsf{NW}, T)$ includes all the messages for any $\mathsf{fv}(V, T)$, though the timestamps may be mismatched. With this definition, our definitions $\mathsf{view}(V, T)$ and $\mathsf{view}(\mathsf{NW}, T)$ respectively are just subsets of the **accepted** messages in $\mathsf{fv}(V, T)$ and $\mathsf{view}(\mathsf{NW}, T)$ respectively.

Now, it is completely possible to describe everything in terms of full views instead of views. However, there are many reasons why we work with views instead of full views in this paper (and why we leave the concept of full views to the appendix). To give a few:

- We can visualize views as a connected tree of blocks and attestations (with edges corresponding to dependencies). To visualize full views, we necessarily have to use a possibly disconnected graph.
- Motivated by real-life concerns, a protocol working with full views would have to contain instructions that differ on how to handle messages a validator $V$ has seen (but shouldn't act on, as it depends on a possibly nonexistent message $V$ has not seen) versus a message a validator has accepted and fully "understands". As an example, a cryptocurrency protocol may have to contain statements of the sort "when a full view sees a transaction about a coin $B$, if the parent blocks of $B$ exist (recursively) and are all signed correctly, then consider the transaction valid." This is very clunky, whereas if parent-child relationships were dependencies then using views abstracts away the recursive-checking logic, and we just need to say "when a view sees a transaction about a coin $B$, consider the transaction valid."
- It is always possible to have a protocol that works on views: if message $B$ depends on message $A$ but we receive $B$ first, then working with views is equivalent to ignoring the existence of message $B$ when making choices as a validator until $A$ (and other dependencies) come.

To summarize: even though the *full view* is somewhat intuitively easier to understand than the *view*, since all the dependencies are met in a view by construction, what the view sees forms a "coherent" state where the validator can reason about any message with no ambiguity, whereas reasoning about the full view will usually come with caveats about checking that all dependencies of the messages involved in the reasoning are met (probably recursively).

## B. The Equivocation Game

Recall that in Section 7, we abstracted a single slot of Gasper into a one-shot game called the *equivocation game*, parametrized by $(\mathcal{V}, a, \epsilon_1, \epsilon_2)$, where $\mathcal{V}$ votes on 2 options $O_1$ and $O_2$. In this Appendix, we give some further details and perform some simulations.

First, we explicitly discuss how time and synchrony conditions are modelled in this game:

1. There is a single time period in this game, formalized as the real interval $[0, 1]$. This interval corresponds to the 1 slot of time in Gasper.

2. Honest validators follow the following protocol: "vote at [your] $t = 0.5$ for the option with more total stake voted in your view; in the case of a tie, vote $O_1$." This protocol corresponds to the instruction in Gasper that attestors for slot $i$ are supposed to attest at time $i + 1/2$ (and tiebreakers are broken by a hash, which is an arbitrary but fixed value).

3. Synchrony model:
   a) We assume that all validators have perfectly synced clocks, but each validator attempting to vote at time $t$ actually votes at "real" time $t + X$ (rounding to inside the interval $[0, 1]$), where $X$ is a uniform random variable with support $[-\epsilon_1, \epsilon_1]$, independently re-sampled for each message. We can think of $\epsilon_1$ as a "timing error" bound, accounting for clock differences, client-side timing issues, etc.
   b) When a validator votes at "real" time $t'$, all other validators obtain the vote at time $t' + a + Y$, where $Y$ is a uniformly distributed variable with support $[-\epsilon_2, \epsilon_2]$. We can think of $a$ as the average delay per message, and $\epsilon_2$ a noise on top of the delay. Combining with the previous point, we have that a validator attempting to vote at time $t$ actually has his/her vote received by another validator at $t + a + X + Y$, where the $X$ and $Y$ are independently re-sampled for each event.

For our analysis in this section, we make the following additional assumptions and notations:

- Every validator has exactly 1 unit of stake. Barring further knowledge (such as empirical data) about validator stake distributions, this is the most intuitive choice for our toy model.

- We define $N_h \geq 2N/3$ to be the total number of honest validators (which is equivalent to their total amount of stake). In reality (and in Section 7), we expect $N_h$ to be a bit bigger at $2N/3 + \epsilon N$ for small $\epsilon$.

- Similarly, we define $N_b \leq N/3$ to be the number of byzantine validators. We use $p = \frac{N_b}{N}$ to be the proportion of byzantine validators.

Finally, for our computer simulations, we additionally set the parameters to $\mathcal{V} = 111$, $N_h = 74$, $N_b = 37$. The total number of validators of 111 is based on

[6], as a heuristic lower bound for making the committees safe. Erring on the conservative side, we pick the maximally pessimistic parameter of $p = 1/3$ fraction of byzantine validators. Having more honest validators than this assumption significantly improves our chances of winning.

## B.1. The Pessimistic Regime - High Latency

This pessimistic regime is a mental experiment to show that under certain conditions fair to both sides, the power of collusion is enough to make the game favor the dishonest side.

We assume that $a$ (the delay for messages from one validator to another) is big compared to $\epsilon_1$ (the error in timing a vote). In this situation, we argue that the dishonest validators can perform the following "smoke bomb attack": at time close to $0.5 - a$, all the dishonest validators vote in a way that the votes are split between the 2 options, for a total of $pN/2$ votes for each option, all with fake timestamps for time close to 0.5.

When this attack activates, from the view of each honest validator, at time 0.5 they will see some random votes from the dishonest validators. As $a$ is big compared to $\epsilon_1$, they almost never see other votes from honest validators. Thus, the dishonest validators and honest validators both end up splitting the vote: the dishonest validators end up splitting $pN$ of the votes, and the honest validators split the remaining $(1 - p)N$ probabilistically.

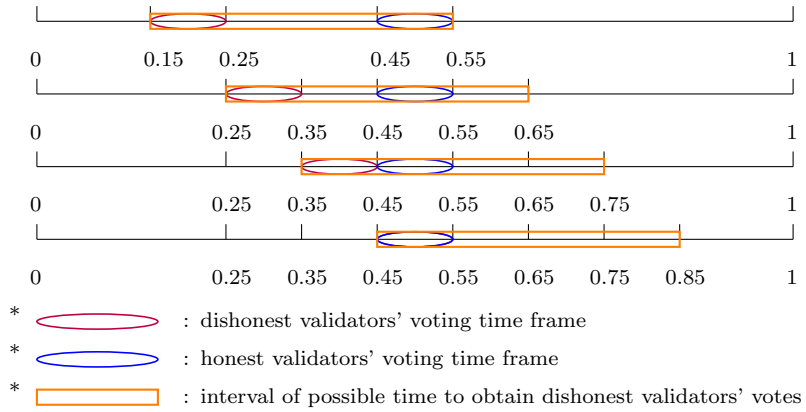
Assuming the honest validators all have 1 stake, the honest validator's votes can be modeled by $(1 - p)N$ Bernoulli trials between the two outcomes, ending up close to a normal distribution around a tie, with a standard deviation proportional to $\sqrt{(1 - p)N}$ votes. In particular, for large $N$, it is very unlikely for one of the options to get anywhere near $2N/3$ stake worth of votes.

**Remark.** As an extension of this example, we can also consider the case where the protocol tiebreaker is "flip a coin," the situation becomes even worse: the dishonest validators can simply wait until the end to vote. Because $a$ is big compared to $\epsilon_1$, the honest validators are still essentially making coin flips since they have not seen any other votes. Our tiebreaker rule (vote for option $O_1$ if there is a tie) solves this problem. The main takeaway here is that these protocols (including attacks on them) are very sensitive to very small implementation details.

In Figure 12, we have four cases that demonstrate different outcomes of the pessimistic regime with $a = 0.15$, $\epsilon_1 = 0.05$, and $\epsilon_2 = 0.15$. This ensures that messages sent at time $t$ are received uniformly randomly in the interval $[t, t+0.3]$. Note that having smaller $\epsilon_2$ would be even more pessimistic, almost ensuring that honest validators cannot see each other's votes.

In Case 1, dishonest validators vote much earlier than honest validators do, so most of the attestations made by the former are visible to the latter when they vote. Thus, the honest validators are likely to vote for the same option. For Cases 2 and 3, dishonest validators vote closer and closer to when the honest validators do, which increases the power of the attack; in Case 3 dishonest validators win at

42% of the time. In Case 4, dishonest validators vote at the same time as honest validators; because of the delay, the dishonest votes do not confuse the honest validators, so honest validators almost always win (because they are likely to vote for the default option $O_1$, barring seeing any votes).



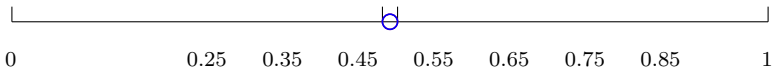

| Dishonest voting time | Win |
|---|---|
| 0.2 | 96% |
| 0.3 | 74% |
| 0.4 | 58% |
| 0.5 | 100% |

Figure 12: Pessimistic regime simulations with $a = 0.15$, $\epsilon_1 = 0.05$, $\epsilon_2 = 0.15$ for the 4 cases above in order. Each row has a different coordinated dishonest voting time, which affects the winning rates of honest validators.

## B.2. The Optimistic Regime - Low Latency

For this regime, we go to the other extreme and assume perfect synchrony ($a = \epsilon_2 = 0$). This means all decisions are immediately propagated to all other validators on the network.

In this game, the moment one choice has a lead (by default, $O_1$), everybody is aware of the lead and all honest validators will vote for the choice in the lead. Thus, the optimal strategy for dishonest validators is to keep the voting exactly at a tie (in which case the next honest validator will vote for $O_1$) or such that $O_2$ has one extra vote (in which case the next validator will vote $O_2$). However, as it takes more votes to get the count to under $O_1$, the dishonest validators really can do no better than as if they voted at the very end in the wrong direction, which still gives the honest validators $(1 - p)N$ votes in the correct direction. In our situation where $p = 1/3$, this gives a total vote differential of $(1 - 2p)N = N/3$, which is good for us. One can see a simulated outcome of this case in Figure 13.




| Dishonest voting time | Win |
|---|---|
| 0.5 | 100% |

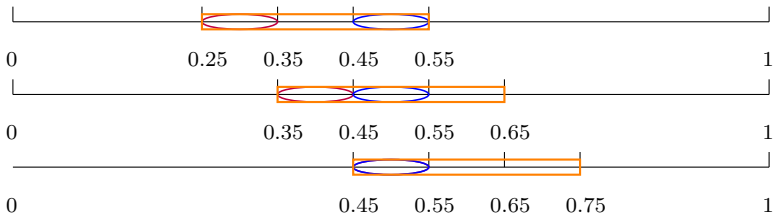
Figure 13: Optimistic regime simulation outcome with $a = 0$, $\epsilon_1 = 0.05$, $\epsilon_2 = 0$ with both dishonest and honest validators voting in the same time frame. Honest validators always win.

## B.3. An Example Inbetween

We make a specific set of assumptions somewhere between the extremes presented in sections B.1 and B.2. These assumptions are fairly arbitrary and hopefully presents a realistic example. It would be good to find principled "bottom truth" parameters somehow (such as after the actual blockchain is implemented).

In Figure 14, we have three cases that demonstrate the outcomes of the inbetween examples with $a = 0.1, \epsilon_1 = 0.05$ and $\epsilon_2 = 0.1$. Thus, $a$ is comparable to $\epsilon_1$ (avoiding the pessimistic case) but fairly big (avoiding the optimistic case). Making $\epsilon_2$ bigger doesn't really change the results much until $\epsilon_2$ becomes around the order of $a$.

In Case 1, some dishonest validators vote earlier than honest validators do and their "smoke bomb attack" has some impact; however, the amount of votes are not distracting enough compared to the amount of votes honest validators are able to vote following the protocol. In case 2, as we earlier discussed in the pessimistic regime, dishonest validators vote closer to when the honest validators do, which boosts the effectiveness of the attack. In Case 3, dishonest validators vote at the same time as honest validators do; as in the pessimistic regime, honest validators still almost always win.




| Dishonest voting time | Win |
|---|---|
| 0.3 | 93% |
| 0.4 | 79% |
| 0.5 | 99% |

Figure 14: Simulation outcomes with $a = 0.1$, $\epsilon_1 = 0.05$, $\epsilon_2 = 0.1$.