The CasperLabs Highway Protocol

Daniel Kane[1], Vlad Zamfir[2], and Andreas Fackler[3]

[1]Computer Science and Engineering Department, UC San Diego
[2]Ethereum Research
[3]CasperLabs LLC

September 2019

Contents

Abstract

We present a concrete consensus algorithm based on [Zam+18] together with a criterion for finality, both proposed by Daniel Kane. This consensus algorithm is live in a partially synchronous network and has variable out-of-protocol finality thresholds each observer can choose independently, thus allowing individual tradeoffs between safety and liveness.

At the heart of the protocol are two design choices that keep it simple and intuitive:

- There are no targeted messages sent only to specific nodes, like e.g. [Mil+16], and no messages that contain a particular choice of other messages that the sender had received earlier, [C+99]'s `NEW-VIEW` message. Instead, all messages are eventually delivered to everyone, and they attest to earlier messages; all nodes see the same, continuously growing graph of messages, similar to [Bai16].
- Decisions are made using simple voting. Liveness is ensured by enforcing a particular structure in the graph that prevents stalemates from persisting.

In section 1 we introduce the basic concepts of messages, justifications and equivocations, and analyze the safety guarantees that can be derived from their structure so that we can determine in a general setting when consensus decisions are final, i.e. under some reasonable assumptions they cannot be reverted.

Section 2 provides an explanation about how these results can be applied to the specific use case of a blockchain.

In section 3, we introduce additional rules that guarantee not only safety, but also liveness; decisions can be relied on and are eventually actually made.

Finally, section 4 puts the abstract consensus algorithm in the context of a distributed proof of stake network, outlines how deposits, rewards and penalties can be handled, and addresses the problem of long range attacks.

1 Messages and Finality

1.1 Justifications

In Casper, every message μ sent by an honest validator v contains attestations of v to messages sent or received before. These earlier messages are referred to as *justifications*, and we say that μ *cites* them.

In theory, this approach could be implemented by including all previous messages in μ. In practice, we include the hashes of the previous messages, omitting redundant ones: if message μ_3 cites μ_2, and μ_2 cites μ_1, then μ_3 does not need to explicitly include μ_1's hash anymore. Nevertheless,

we consider μ_1 a justification of μ_3. Conversely, if μ_1 is a justification of μ_2, all justifications of μ_1 are also justifications of μ_2. In summary:

Definition 1. *Given a set $\mathcal{M}$ of* messages*, a* justification function *is a function $J : \mathcal{M} \to \mathcal{P}(\mathcal{M})$, such that:*

- *For every $\mu \in \mathcal{M}$, $J(\mu)$ is finite.*
- *Whenever $\mu \in J(\lambda)$, then $J(\mu) \subseteq J(\lambda)$.*

The elements of $J(\mu)$ are the justifications *of μ. We also write $\mu < \lambda$ for $\mu \in J(\lambda)$, and say that λ* cites *μ.*

A finite set $\sigma \subseteq \mathcal{M}$ is called a protocol state *if it is closed under J, i.e. if $J(\mu) \subseteq \sigma$ for every $\mu \in \sigma$. We denote the set of all protocol states as $\Sigma_{\mathcal{M}}$, or just Σ, if $\mathcal{M}$ is clear from the context.*

The relation $<$ is a well-founded[1] strict partial order[2] on $\mathcal{M}$. We use the usual notation for partial orders, e.g. $\mu \leq \lambda$ means $\mu \in J(\lambda) \cup \{\lambda\}$. We also say *$\lambda$ sees μ* if $\mu \leq \lambda$.

Note that $J(\mu)$ and $J(\mu) \cup \{\mu\}$ are protocol states for every message μ.

In a network, every node maintains its own current protocol state that includes all messages it sent and received so far — with the exception of messages μ for which some justification $\lambda \in J(\mu)$ has not yet been received. Those are kept in a temporary storage while the node requests λ from its peers. That means a node's protocol state is monotonically increasing; if it has state σ before it has state τ, then $\sigma \subseteq \tau$.

Moreover, we assume that all correct nodes will continuously sync with each other and exchange all messages they know of. So, whenever two honest nodes are in states σ and σ', they will eventually reach states τ and τ' such that $\sigma \cup \sigma' \subseteq \tau$ and $\sigma \cup \sigma' \subseteq \tau'$.

1.2 Equivocations

Messages are cryptographically signed by and contain the name (or ID) of their sender. Together with the justifications, this allows us to enforce many aspects of correct behavior on the protocol level, and declare as invalid the messages that do not follow those behavior rules.

Unless otherwise specified, we will implicitly assume a fixed set $\mathcal{M}$ of messages, together with a justification function J, and a function $S : \mathcal{M} \to \mathcal{V}$ that assigns its sender $S(\mu)$ to each message μ from some set $\mathcal{V}$ of *validators*.

Definition 2. *Given a validator $v \in \mathcal{V}$ and a protocol state σ, we call the set* $\mathrm{Swim}_v(\sigma) = \{\mu \in \sigma \mid S(\mu) = v\}$ *of all of v's messages in σ, the* swimlane of v in σ.

[1] i.e. every nonempty set has a minimal element. This is true because for every message μ, the set $\{\lambda \mid \lambda < \mu\} = J(\mu)$ is finite, so there are no infinite descending sequences.

[2] It is irreflexive because $\mu \notin J(\mu)$ and transitive because $J(\mu) \subseteq J(\lambda)$ whenever $\mu < \lambda$.

When creating a new message μ, a validator v is expected to always include all of v's previous messages in $J(\mu)$. An honest validator's swimlane $\text{Swim}_v(\sigma) = \{\mu_1, \mu_2, \mu_3, \ldots\}$ will always be totally ordered chronologically: $\mu_1 < \mu_2 < \mu_3 < \ldots$. Violating this rule means producing two messages that don't see each other:

Definition 3. *A pair of messages μ and λ is an* equivocation*, if $S(\mu) = S(\lambda)$ and $\mu \not\leq \lambda$ and $\lambda \not\leq \mu$. The sender $v = S(\mu)$ is an* equivocator*. For every protocol state σ, let*

$$E(\sigma) = \{v \in \mathcal{V} \mid \exists \mu, \lambda \in \text{Swim}_v(\sigma)\ \mu \not\leq \lambda \wedge \lambda \not\leq \mu\}$$

be the set of validators who produced an equivocation that is contained in σ.

Since honest nodes will eventually exchange all messages they know of, any equivocation will eventually become known to all honest nodes. And, if $\sigma \subseteq \tau$, then $E(\sigma) \subseteq E(\tau)$.

1.3 Votes and Weights

The protocol will support a changing set of active validators with different weights, but we will study this aspect later. For now, we consider a fixed map:

Definition 4. *A* validator map *is a map $\mathbb{w} : \mathcal{V} \to \mathbb{R}_{\geq 0}$, assigning to each validator $v \in \mathcal{V}$ a* weight *$\mathbb{w}(v)$, such that at least one, but only finitely many validators are assigned a positive weight*[3].

The weight *of a subset $V \subseteq \mathcal{V}$ of validators is the sum of its members' weights: $\mathbb{w}(V) = \sum_{v \in V} \mathbb{w}(v)$, and the* weight *of a set of messages $X \subseteq \mathcal{M}$ is the sum of the weight of their senders (counting each sender only once, even if they have multiple messages):*

$$\mathbb{w}(X) = \mathbb{w}(S(X)) \text{ , where } S(X) = \{S(\mu) \mid \mu \in X\}$$

A central part of Casper's design, messages count as votes for consensus values, and validators are required to follow the plurality of the non-equivocating validators' latest votes. However, in practice every message will count as a vote for several different things simultaneously. Specifically, in the case of blockchain a message containing a block is always also a vote for all of that block's ancestors.

To capture this notion, we will consider functions $f : \mathcal{M} \to C \cup \{\mathbb{0}\}$, assigning to each message μ a value $f(\mu) \in C$ that the message is voting for, or a special value $\mathbb{0} \notin C$ if the message does not carry a vote, i.e. counts as an abstention. To tally the votes, we will need to look at each validator's latest non-abstention message.

[3] To simplify the notation, instead of changing the set $\mathcal{V}$ itself, we will only change the map w, and assign 0 to all validators who are not currently active. So, conceptually $\mathcal{V}$ is the set of all *potential* validators and can even be infinite. In an implementation, w should be represented as the finite map containing only the non-zero entries.

Definition 5. *The set of* latest honest messages with property[4] *p*

$$L_p(\sigma) = \{\mu \in \sigma \mid S(\mu) \notin E(\sigma) \wedge p(\mu) \wedge \forall \lambda \in \mathrm{Swim}_{S(\mu)}(\sigma)\ (\neg p(\lambda)\ \vee\ \lambda \leq \mu)\}$$

*contains the latest messages with property p of all non-equivocating validators. We also call messages with property p p-*messages.

For a message μ, we define $L_p(\mu) = L_p(J(\mu))$ for brevity, and we write L without a subscript for the latest honest messages with the always-true property, i.e.:

$$L(\sigma) = \{\mu \in \sigma \mid S(\mu) \notin E(\sigma) \wedge \forall \lambda \in \mathrm{Swim}_{S(\mu)}(\sigma)\ \lambda \leq \mu\}$$

Recall that non-equivocating validators' swimlanes are totally ordered, so $L(\sigma)$ contains exactly the maxima of their swimlanes.

Given a voting function $f : \mathcal{M} \to C \cup \{\mathbb{0}\}$ with a totally ordered set C, let

$$L_f(\sigma) = \{\mu \in \sigma \mid S(\mu) \notin E(\sigma)\ \wedge\ f(\mu) \neq \mathbb{0}\ \wedge\ \forall \lambda \in \mathrm{Swim}_{S(\mu)}(\sigma)\ (f(\lambda) = \mathbb{0}\ \vee\ \lambda \leq \mu)\},$$

i.e. $L_f = L_p$ for the property $p(\mu) \Leftrightarrow f(\mu) \neq \mathbb{0}$. Let

$$W_f(c, \sigma) = \mathbb{w}(\{\mu \in L_f(\sigma) \mid f(\mu) = c\})$$

for $c \in C$: the total weight of the non-equivocating validators whose latest non-$\mathbb{0}$ vote was for c.

f is $\mathbb{w}$-plurality-driven *if for every message $\mu \in \mathcal{M}$ with $f(\mu) \neq \mathbb{0}$:*

$$f(\mu) = \min\left(\arg\max_{c \in C} W_f(c, J(\mu))\right)$$

In words: f is $\mathbb{w}$-plurality-driven, if $f(\mu)$ is always the value $c \in C$ which the highest total weight of non-equivocating validators voted for in their latest non-$\mathbb{0}$ messages, using the total order on C as a tie-breaker.

In particular, if more than half of the votes (by weight) in $L_f(\mu)$ are for a value c, and $f(\mu) \neq \mathbb{0}$, μ must also be: $f(\mu) = c$. So if f is $\mathbb{w}$-plurality-driven, the property $f(\mu) = c$ is $\mathbb{w}$-majority-driven relative to $f(\mu) \neq \mathbb{0}$, in the following sense:

Definition 6. *A property of messages p is* $\mathbb{w}$-majority-driven relative to *a property q if for every valid message μ:*

$$w(\{\lambda \in L_q(\mu) \mid p(\lambda)\}) > \frac{1}{2}\mathbb{w}(L_q(\mu)) \quad \wedge \quad q(\mu) \quad \Rightarrow \quad p(\mu)$$

I.e. whenever μ has property q and a strict majority of latest honest messages in $J(\mu)$ with property q has property p, then μ also does.

[4] Properties can be represented as a subset $p \subseteq \mathcal{M}$. We say that *p applies to μ*, or just $p(\mu)$, if $\mu \in p$.

p is $\mathbb{w}$-majority-driven *if it is* $\mathbb{w}$*-majority-driven relative to the always-true property, i.e. if for every valid message* μ*:*

$$w(\{\lambda \in L(\mu) \mid p(\lambda)\}) > \frac{1}{2}\mathbb{w}(L(\mu)) \quad \Rightarrow \quad p(\mu)$$

I.e. whenever a strict majority of latest honest messages in $J(\mu)$ *has property p, then* μ *also does.*

These definitions are motivated by the blockchain case where we will prove that for every block b, "the child c of b such that the message votes for (a descendant of) c" is plurality-driven. This will imply that if c is a child of b, the property "the message votes for c" is majority-driven relative to "the message votes for b". See section 2 for details. We will examine conditions under which majority-driven properties become finalized such that all future messages have them.

Lemma 1. *If* $f : \mathcal{M} \to C \cup \{\mathbb{0}\}$ *is* $\mathbb{w}$*-plurality-driven,* $f(\lambda) \neq \mathbb{0}$*,* $f(\mu) \neq \mathbb{0}$*, and* $J(\mu) = J(\lambda) \cup \{\lambda\}$*, then* $f(\mu) = f(\lambda)$*.*

We will later try to construct situations like this, where all messages directly follow some "leader" message λ. The lemma shows that in such situations, agreeing votes are enforced.

Proof. The only added message in $J(\mu)$ compared to $J(\lambda)$ is λ itself, which is comparable to all other elements of $J(\mu)$, and hence cannot be part of an equivocation. So, $E(J(\mu)) = E(J(\lambda))$.

If $S(\lambda) \in E(J(\lambda))$, then $L_f(J(\mu)) = L_f(J(\lambda))$, otherwise $L_f(J(\mu)) = L_f(J(\lambda)) \cup \{\lambda\} \setminus X$, where X is the set containing $S(\lambda)$'s most recent non-$\mathbb{0}$ message in $J(\lambda)$, or $\emptyset$ if there is none.

So all that changed in $L_f(J(\mu))$ compared to $L_f(J(\lambda))$ is that possibly a vote for $f(\lambda)$ was added, and if so, possibly a vote with the same weight for some $c \in C$ was removed. Thus $W_f(f(\lambda), J(\mu)) \geq W_f(f(\lambda), J(\lambda))$, and $W_f(c, J(\mu)) \leq W_f(c, J(\lambda))$ for all other $c \neq f(\lambda)$.

That means $\arg\max_{c \in C} W_f(c, J(\mu))$ is a subset of $\arg\max_{c \in C} W_f(c, J(\lambda))$ and still contains $f(\lambda)$. Since $f(\lambda)$ was the minimum of the latter, it is also the minimum of the former, hence $f(\mu) = f(\lambda)$. □

1.4 Summits

Definition 7. *A* summit $S = ((C_i)_{i \in \mathbb{N}}, \sigma, p, p', q)$ *is a collection of the following data:*

- *A descending sequence of sets of validators* $\mathcal{V} \supseteq C_0 \supseteq C_1 \supseteq C_2 \supseteq \ldots$ *called* committees*.*
- *A protocol state* σ *containing no equivocation by a member of* C_0*, i.e.* $C_0 \cap E(\sigma) = \emptyset$*.*
- *A property p that is* $\mathbb{w}$*-majority-driven relative to* p'*.*
- *A quorum size* $q > 0$*.*

Given a summit $\mathcal{S}$ and a p'-message $\mu \in \sigma$ with $S(\mu) \in C_0$, we say that μ is of level at least 0 in $\mathcal{S}$ *if $p(\lambda)$ for every p'-message $\lambda \in \mathrm{Swim}_{S(\mu)}(\sigma)$ with $\lambda \geq \mu$, i.e. if μ itself has property p and every later p'-message in σ by the same validator also does.*

For integers $k > 0$, we say that a p'-message $\mu \in \sigma$ with $S(\mu) \in C_k$ is of level at least k in $\mathcal{S}$ *if and only if μ is level at least 0 in $\mathcal{S}$ and there is a subset $V \subseteq C_k$ so that:*

- *For each $v \in V$ there is a $\lambda \leq \mu$ with $S(\lambda) = v$ of level at least $k-1$ in $\mathcal{S}$.*
- *The sum of the weights of the validators in V is at least q, i.e. $\mathbb{w}(V) \geq q$.*

We say that $\mathcal{S}$ is valid *if for every k and every $v \in C_k$ there is a message $\mu \in \sigma$ with $S(\mu) = v$ of level at least k in $\mathcal{S}$.*

Note that committees are allowed to be empty, and usually all but a finite number of committees will be. We are ready to prove that this implies a finality condition:

Theorem 1. *Suppose that there is a valid summit $\mathcal{S} = ((C_i)_{i\in\mathbb{N}}, \sigma, p, p', q)$ with $q = \mathbb{w}(\mathcal{V})/2 + t$ for some $t > 0$. Then, if there is a p'-message μ with $\neg p(\mu)$ that sees a message of level at least k of $\mathcal{S}$, the sum of the weights of members of C_1 who equivocate must be at least $2t(1-2^{-k})$. In fact, there is a subset of C_1 of weight $2t(1-2^{-k})$, each of whose members is*

1. *in $E(J(\mu))$,*
2. *or in $E(\sigma \cup J(\mu))$ and has a message $\epsilon \in L_{p'}(J(\mu))$ coming after a level-at-least-0 message from σ with $\neg p(\epsilon)$.*

Proof. We proceed by induction on k. We note that the $k = 0$ case follows trivially, as it is always the case that a set of validators of C_1 of total weight at least $2t(1-2^{-0}) = 0$ have equivocated.

We assume that our theorem statement holds for $k-1$. Suppose that there is some p'-message μ with $\neg p(\mu)$ that sees a message ν in σ of level at least k. Then we may assume that we have a minimal[5] such μ. In particular, we assume that no p'-message in $J(\mu)$ with $\neg p(\mu)$ sees a message in σ of level at least k.

We will attempt to show that a set of members of C_1 of total weight at least $2t(1-2^{-k})$ must have equivocated.

We note that as ν is level at least k, there is a set $V \subseteq C_k$ of validators of total weight at least $\mathbb{w}(\mathcal{V})/2+t$ where for every $v \in V$, ν sees a message by v of level at least $k-1$ in σ. All of these p'-messages have property p. Normally, a set of p'-messages of total weight more than $\mathbb{w}(\mathcal{V})/2$ would force μ to vote for p, since p is $\mathbb{w}$-majority-driven relative to p'. However, some of these validators might be seen by μ as either equivocating or changing their vote. Let $a = \mathbb{w}(E(J(\mu)) \cap V)$ be the

[5] I.e. if there is a p'-message $\mu' < \mu$ with $\neg p(\mu')$ that also sees a message ν' in σ of level k, then we replace μ with μ' and ν with ν'. Note that since $J(\mu) \supseteq J(\mu')$ that $E(J(\mu)) \supseteq E(J(\mu'))$ and that any validator with an equivocation in $\sigma \cup J(\mu')$ has one in $\sigma \cup J(\mu)$, and the existence of an ϵ as in 2 holds as well.

sum of the weights of validators in V that μ sees equivocate. Let $b = \mathbb{w}(\{\kappa \in L_{p'}(\mu) \mid S(\kappa) \in V \wedge \neg p(\kappa)\}$ be the total weight of the validators in V who do not have equivocations cited by μ but whose latest p'-message cited by μ does not have property p. In order for μ not to be forced to have p by the majority-driven rule, it must be the case that $\mathbb{w}(\mathcal{V})/2 + t - a - b \leq (\mathbb{w}(\mathcal{V}) - a)/2$, or equivalently, $a + 2b \geq 2t$.

We note that if $b = 0$, we are already done as in this case $a \geq 2t$. Thus, we may assume that $b > 0$. This means that there is some $\xi \in L_{p'}(\mu)$ with $S(\xi) \in V$ and $\neg p(\xi)$. Since ξ is the most recent p'-message, it must see a message of level at least $k - 1$ by $S(\xi)$.

Let κ be a *minimal* p'-message in $J(\mu)$ with $S(\kappa) \in V \setminus E(J(\mu))$ and $\neg p(\kappa)$ that sees a message of level at least $k - 1$ by $S(\kappa)$.

κ cites a message of level at least $k - 1$ but $\neg p(\kappa)$, so the inductive hypothesis implies that there is a set $W \subseteq C_1$ of validators of weight $\mathbb{w}(W) \geq 2t(1 - 2^{-k+1})$, each of which has one of the following properties:

- It is in $E(J(\kappa))$. Then, it is also in $E(J(\mu))$, since $\kappa \leq \mu$.
- It has a p'-message $\epsilon \in L_{p'}(J(\kappa))$ with $\neg p(\epsilon)$ coming after a level-at-least-0 message of σ. By the minimality of κ, ϵ cannot cite a level $k - 1$ message. Thus, if $S(\epsilon)$ sent a message ϵ' citing a level $k - 1$ message, ϵ and ϵ' would be an equivocation. And, if μ sees both, this would cause ϵ to contribute to a.

Thus, none of the members of W contribute to b.

Let c be the weight of validators in C_1 which have equivocations in $\sigma \cup J(\mu)$ who do not contribute to a or b but have a p'-message $\epsilon \in J(\mu)$ coming after some level-at-least-0 message of σ with $\neg p(\epsilon)$.

Note that in the above, validators in the first case contribute to a, and validators in the second contribute to c.

Therefore, we have that $a + c \geq 2t(1 - 2^{-k+1})$. Combining this with $a + 2b \geq 2t$, we have that

$$a + b + c \geq (2a + 2b + c)/2 \geq (a + 2b)/2 + (a + c)/2 \geq 2t/2 + 2t(1 - 2^{-k+1})/2 = 2t(1 - 2^{-k}).$$

The validators contributing to a, b and c are clearly disjoint. Those contributing to a have equivocations in $J(\mu)$ and thus satisfy 1. Those contributing to c have equivocations and p'-messages that are post-level-at-least-0 and contradict p, thus satisfying 2. For those contributing to b, we note that by the minimality of μ, their vote for $\neg p$ could not have contained in its justification a message of level at least k, even the first message ζ of level at least k that this validator must have cast. However, since this is their most recent p'-message it must be an equivocation against ζ. Therefore, this equivocation would be contained in $\sigma \cup J(\mu)$. Furthermore, these validators will have cast post-level-0 p'-messages that contradict p, and hence they satisfy 2.

Thus, the validators contributing to a, b and c have total weight of at least $2t(1 - 2^{-k})$, and all satisfy either 1 or 2. □

Thus, once a validator has such a summit in their local protocol state σ, they know that eventually every honest validator will also see all of σ, and will only produce p'-messages with property p from then on, unless a weight of at least $2t(1 - 2^{-k})$ validators equivocate. They can therefore consider p to be *finalized relative to* p', with fault-tolerance threshold $2t(1 - 2^{-k})$.

Note that if we have at least a $2\mathbb{w}()/3$-weight of honest validators, we could reasonably expect all of them to eventually contribute to a summit. If they persisted for enough rounds to achieve a high k, this would allow them to achieve a fault tolerance threshold of nearly $\mathbb{w}()/3$. Thus, ignoring liveness considerations for the moment, we could hope that our system is secure against a third of the validators being faulty. In fact, in practice we might expect to be able to obtain summits with weight close to $\mathbb{w}(\mathcal{V})$ if either there were few faulty validators at the time, or the faulty validators did not behave unusually. In this case, we might hope to obtain a fault-tolerance threshold of nearly everything.

1.5 Computational Efficacy of the Finality Criterion

Theorem 2. *Given a protocol state σ, properties p and p' and a quorum size $q > 0$, there exists a maximal valid summit $((C_i), \sigma, p, p', q)$ in the sense that for every valid summit $((\tilde{C}_i)_i, \sigma, p, p', q)$, $\tilde{C}_i \subseteq C_i$ for all i.*

Furthermore, there exists a polynomial time algorithm to compute (C_i).

Proof. We show that for each $k \geq 0$, there is a unique, easily computable valid summit

$$((C_0, \dots, C_k, \emptyset, \emptyset, \dots), \sigma, p, p', q)$$

so that for any valid summit $((\tilde{C}_i)_{i\in\mathbb{N}}, \sigma, p, p', q)$, we have $\tilde{C}_i \subseteq C_i$ for all $i \leq k$, and that every p'-message that has level at least i in the latter, also has level at least i in the former.

The decreasing sequence $C_0 \supseteq C_1 \supseteq \dots$ of finite sets must eventually be constant, and since each requirement in the definition of a valid summit affects only finitely many elements of it, this proves that $((C_i), \sigma, p, p', q)$ is also a valid summit and a maximal one.

For $k = 0$, we let C_0 be the set of all validators $v \notin E(\sigma)$ whose latest p'-message has property p. Since every validator in $\tilde{C}_0$ must have at least one level ≥ 0 message, it is clear that $\tilde{C}_0 \subseteq C_0$.

Now assume we have constructed $C_0, \dots, C_k$ so that they define a valid maximal summit up to k. Let $D_0 = C_k$, and D_{i+1} the set of all $v \in D_i$ such that there is a p'-message $\mu \in \sigma$ with $S(\mu) = v$ and $\mathbb{w}(\{\lambda \leq \mu \mid S(\lambda) \in D_i \text{and } \lambda \text{ is level } k\}) \geq q$. Then since $D_0 \supseteq D_1 \supseteq \dots$, there is an i with $D_i = D_{i+1}$, and we set $C_{k+1} = D_i$.

We prove by induction on i that $\tilde{C}_{k+1} \subseteq D_i$. For $i = 0$ this is clear since we must have $\tilde{C}_{k+1} \subseteq \tilde{C}_k \subseteq C_k = D_0$. For $i > 0$, assume $\tilde{C}_{k+1} \subseteq D_{i-1}$. Every $v \in \tilde{C}_{k+1}$ has a p'-message of level $k + 1$, i.e. one that sees a set of level k messages of weight $\geq q$ by members of $\tilde{C}_{k+1} \subseteq D_{i-1}$. Thus $v \in D_i$. This completes the inductive step and shows that $\tilde{C}_{k+1} \subseteq D_i$ for all i, and thus $\tilde{C}_{k+1} \subseteq C_{k+1}$.

It follows from the definition that every message that is level $\geq k$ in $((\tilde{C}_i)_{i\in\mathbb{N}}, \sigma, p, p', q)$ is also level $\geq k$ in $((C_i)_{i\in\mathbb{N}}, \sigma, p, p', q)$. This completes our inductive step and the proof. □

2 Blockchain

2.1 Blocks

In a smart contract platform, distributed ledger, or replicated state machine, the actual value that the consensus algorithm tries to reach agreement on is an ever-growing list of transactions. In a *blockchain*, that list is divided into *blocks*, each of which refers to its predecessor, its *parent*, by hash. The first one, the *genesis block*, is part of the protocol definition, or network specification.

The set $\mathcal{B}$ of blocks is therefore a tree, with a unique finite path leading from each block to the root: the genesis block g. We write $\mathrm{Prev}(b)$ for b's parent, so $\mathrm{Prev} : \mathcal{B} \setminus \{g\} \to \mathcal{B}$. b_1 is an *ancestor* of b_2, and b_2 a *descendant* of b_1, and write $b_1 \leq b_2$ if there is a sequence $b_1 = c_1, \ldots, c_n = b_2$, such that $b_k = \mathrm{Prev}(b_{k+1})$ for all $k < n$. If $n > 1$, i.e. $b_1 \neq b_2$, we say *strict ancestor* resp. *strict descendant*, and $b_1 < b_2$.

We recursively define the *block height* as the length of the path back to genesis:

- $H(g) = 0$,
- $H(b) = 1 + H(\mathrm{Prev}(b))$ for $b \neq g$.

Every block with height $\geq k$ has a unique ancestor with height k:

- $A_k(b) = \mathbb{0}$ if $k > H(b)$
- $A_k(b) = b$ if $k = H(b)$
- $A_k(b) = A_k(\mathrm{Prev}(b))$ if $k < H(b)$

In Casper, every message represents a vote for a block: either one that was created earlier, or a new child of such a block. So there is a map $B : \mathcal{M} \to \mathcal{B} \setminus \{g\}$ that assigns to each message the block that it votes for such that $\mathrm{Prev}(B(\mu)) \in \{g\} \cup \{B(\lambda) \mid \lambda < \mu\}$ for all $\mu \in \mathcal{M}$.

If a message μ votes for a pre-existing block, i.e. if there is a $\lambda < \mu$ with $B(\lambda) = B(\mu)$, then μ is called a *ballot*[6]. Otherwise, μ is called a *block message*.

For a block b, let p_b be the message property defined by $p_b(\mu) \Leftrightarrow b < B(\mu)$, i.e. a message has property p_b if it carries a block that is a strict descendant of b.

[6]Since ballots "contain" a block that is already known from one of their justifications, the block should not be sent along again, of course. In the implementation it suffices if the ballot message contains only the hash of the block that it votes for.

2.2 The GHOST Rule

The set of validators and their weights can be managed "on-chain" by a smart contract. Every block b is associated with a particular validator set that corresponds to the contract's state after b:

Let $\mathbb{w}_b : \mathcal{V} \to \mathbb{R}_{\geq 0}$ assign to each block $b \in \mathcal{B}$ and validator $v \in \mathcal{V}$ a *weight* $\mathbb{w}_b(v)$. A validator v is called *active after* b, if $\mathbb{w}_b(v) > 0$. We assume that after every b, there is at least one, but only finitely many active validators.

For theoretical purposes, assume that the Hash $: \mathcal{B} \to \mathbb{N}$ function is injective, i.e. there are no hash collisions.

The GHOST rule is the requirement that for every message $\mu \in \mathcal{M}$, either $B(\mu)$ or $\text{Prev}(B(\mu))$ is the GHOST choice of $J(\mu)$, where the *GHOST choice* of a protocol state σ is the unique block a, such that:

- a is a maximal element of $\{g\} \cup \{B(\mu) \mid \mu \in \sigma\}$.
- For every $k < H(a)$,

$$A_{k+1}(a) = \min\left(\arg\max_{c \in \mathcal{B}} \mathbb{w}_b(\{\mu \in L_{p_b}(\sigma) \mid c = A_{k+1}(B(\mu))\})\right),$$

  where $b = A_k(a)$ and the minimum is taken with respect to the block hash.

In other words, the GHOST choice of σ is the block that is reached by starting from $b = g$, and recursively replacing b with the child c of b with maximal $\mathbb{w}_b(\{\mu \in L_{p_b}(\sigma) \mid c \leq B(\mu)\})$, using the block hash as a tie-breaker. The rule says that your message must vote for a block b that is either the GHOST choice itself, or its child.

Note that the GHOST rule can be verified using only the message μ itself and its justifications, but no other messages (which although they may exist, we don't need to use). Messages violating the rule can be detected before a node would insert them into its protocol state; they can simply be treated as invalid and discarded.

Proposition 1. *Assume that the GHOST rule holds for all messages in $\mathcal{M}$. Given a block c with height $k+1$ and parent $b = \text{Prev}(c)$:*

1. *The property $q(\mu) \Leftrightarrow c \leq B(\mu)$ is $\mathbb{w}_b$-majority-driven relative to p_b.*
2. *The function $f : \mathcal{M} \to \mathcal{B}$, defined as follows, is $\mathbb{w}_b$-plurality-driven (where as a tie-breaker, blocks are ordered by hash): $f(\mu) = A_{k+1}(B(\mu))$ if $b < B(\mu)$, otherwise $f(\mu) = \mathbb{0}$.*

Proof. For point 1, assume that $w(\{\lambda \in L_{p_b}(\mu) \mid q(\lambda)\}) > \frac{1}{2}\mathbb{w}(L_{p_b}(\mu))$ and $p_b(\mu)$. In particular $b < B(\mu)$, so $A_{k+1}(B(\mu)) \neq \mathbb{0}$. The GHOST rule implies that

$$A_{k+1}(B(\mu)) = \min\left(\arg\max_{c' \in \mathcal{B}} \mathbb{w}_b(\{\mu \in L_{p_b}(\sigma) \mid c' = A_{k+1}(B(\mu))\})\right),$$

and the only element in that arg max can be c. So $A_{k+1}(B(\mu)) = c$ and thus $q(\mu)$.

For point 2, assume $f(\mu) \neq \mathbb{0}$, i.e. $b < B(\mu)$. Since $f(\mu) = A_{k+1}(B(\mu))$, the second GHOST rule point implies that $f(\mu)$ is the block c with the minimal hash which maximizes $\mathbb{w}_b(\{\lambda \in L_{p_b}(J(\mu)) \mid c = A_{k+1}(B(\lambda))\})$. But $L_{p_b} = L_f$ and for $\lambda \in L_f(J(\mu))$, $c = A_{k+1}(B(\lambda))$ is equivalent to $f(\lambda) = c$. So $f(\mu)$ actually maximizes $W_f(c, J(\mu))$. Hence f is $\mathbb{w}_b$-plurality-driven. □

3 Liveness

3.1 Leaders and Ticks

We measure time in milliseconds since the epoch[7]. An integer timestamp is called a *tick*. We pseudorandomly assign one of the active validators $\mathcal{L}(i) \in \mathcal{V}$ to each tick[8] i as the tick's *leader*.

We assume that messages have timestamps: $T : \mathcal{M} \to \mathbb{R}$, i.e. $T(\mu)$ is the time when message μ was sent[9].

Each validator v maintains a private parameter $n_v(i) \in \mathbb{N}$ that is updated periodically. $n_v : \mathbb{N} \to \mathbb{N}$ maps each tick number to the parameter value. We will give strategies to select n_v below, but we always assume that $n_v(i) = n_v(i-1)$ unless i is a multiple of both $2^{n_v(i)}$ and $2^{n_v(i-1)}$. In other words, we assume that the parameters are kept constant for time windows of $2^{n_v(i)}$ ticks, from j to $j + 2^{n_v(i)} - 1$, where $j \leq i$ is maximal such that it divides $2^{n_v(i)}$. Furthermore, we fix a parameter[10] $0 < R < 1$.

A validator v will create and gossip new messages as follows. Let i be the most recent tick, and j the most recent tick divisible by $2^{n_v(i)}$. v creates a message:

- at time j if v is the leader of j,
- if v is not j's leader, as soon as it receives the tick-j-message λ from j's leader, and
- unconditionally at time $j + R \cdot 2^{n_v(i)}$.

In the first and third case, v includes as justifications its full protocol state σ. In the second case, v includes as justifications only $J(\lambda) \cup \{\lambda\}$ and v's own previous message and its justifications.

So in each round lasting $2^{n_v(i)}$ ticks, first the "round leader", i.e. the leader of the round's first tick, sends a message λ to everyone. Other validators send a message to everyone as soon as they received the leader's message. After a fraction R of the round has passed, every validator sends a message to everyone again. The intention is for the first message to confirm λ's vote and

[7] Unix time: number of milliseconds since the beginning of 1970, UTC

[8] This needs to be done in a secure way; more on this and validator set changes later.

[9] Millisecond resolution is sufficient here, but to keep notation simple, we assume exact timestamps.

[10] A good choice is probably $R = 2/3$, but this should be determined experimentally.

become a level-0 message, and the second one to confirm a sufficient number of first messages, and become a level-1 message, thus forming a summit.

Lemma 2. *Let $f : \mathcal{M} \to \mathcal{C} \cup \{\mathbb{0}\}$ be plurality-driven, and $c \in \mathcal{C}$.*

Let $H \subseteq \mathcal{V}$ be a set of honest validators with $\mathbb{w}(H) > \mathbb{w}(\mathcal{V})/2$, and let j be a tick that is divisible by $2^{n_{\max}}$, where $n_{\max} = \max_{v \in H} n_v(j)$. Let $n_{\min} = \min_{v \in H} n_v(j)$. Assume that j's leader l is in H, and that there is an $R' < R$ such that:

1. *Every message μ with $S(\mu) \in H$ and $T(\mu) = j - (1 - R) \cdot 2^{n_{S(\mu)}(j-1)}$ reaches l by tick j.*
2. *The unique message λ with $S(\lambda) = l$ and $T(\lambda) = j$ reaches every $v \in H$ by time $j + R'2^{n_{\min}}$.*
3. *For every $v \in H$, every message μ with $S(\mu) \in H$ and $j \leq T(\mu) \leq j + R' \cdot 2^{n_v}(j)$ reaches v by time $j + R \cdot 2^{n_v(j)}$.*
4. *$f(\mu) \neq \mathbb{0}$ for every $\mu \geq \lambda$ with $\mu \in H$.*

Then after tick $j + R \cdot 2^{n_{\max}}$, there is a level-1 summit with $q = \mathbb{w}(H)$ consisting of all members of H, finalizing the value $c = f(\lambda)$ relative to $f \neq \mathbb{0}$.

Proof. Every $v \in H \setminus \{l\}$ receives λ before $j + R' \cdot 2^{n_{\min}} \leq j + R' \cdot 2^{n_v(j)}$, so the first message μ_v^0 that v sends after tick j has $J(\mu_v^0) = J(\lambda) \cup \{\lambda\} \cup J(\mu') \cup \{\mu'\}$, where μ' is v's previous message.

Since μ' was sent at $j - (1 - R) \cdot 2^{n_v(j-1)}$, it reached λ by tick j, so $\mu' < \lambda$ and therefore $J(\mu_v^0) = J(\lambda) \cup \{\lambda\}$. By Lemma 1, this implies that $f(\mu_v^0) = c$.

For l itself, we define $\mu_l^0 = \lambda$, so $f(\mu_l^0) = c$ is also true.

Since $T(\mu_v^0) \leq j + R' \cdot 2^{n_w}(j)$ for every $w \in H$, μ_v^0 reaches every $w \in H$ before w creates its second message μ_w^1 after tick j.

So for all $v, w \in H$, $\mu_w^1 > \mu_v^0$.

We will prove that the μ_v^0 are level-0, and the μ_v^1 are level-1 messages of a summit with quorum size $\mathbb{w}(H)$ for the property that f has value c. We already know that each μ_v^1 can see all messages μ_w^0, which have total weight $\mathbb{w}(H)$, so all we have to show is that the latter are level-0 messages, i.e. that no validator $v \in H$ changes away from the value c after μ_v^0.

We prove this by induction on $<$. So let κ be a message by $v \in H$ and $\kappa \geq \mu_v^0$:

If $\kappa = \mu_v^0$, we already know that $f(\kappa) = c$.

So assume now that $\kappa > \mu_v^0$. Then $\kappa \geq \mu_v^1$ and hence $\mu_w^0 < \kappa$ for every $w \in H$. So for every $w \in H$, w's message $\xi \in L_f(\kappa)$ is at least $\xi \geq \mu_w^0$. By the induction hypothesis, $f(\xi) = c$. Therefore, in $L_f(\kappa)$ a strict majority $\geq \mathbb{w}(H)$ is voting for c, so since f is plurality-driven, $f(\kappa) = c$. □

In a blockchain context, leader messages λ (first bullet point) should always carry new blocks. To minimize time to finalization, all other messages μ should be required to be ballots, and just vote

for the most recent leader block: $B(\mu) = B(\lambda)$. Thus whenever the lemma's requirements are satisfied in any round, the full blockchain up to the most recent leader's block becomes finalized, because by proposition 1, the function $\mu \mapsto A_{H(B(\lambda))}(B(\mu))$ is plurality-driven.

That means, to have a live blockchain we just need to make sure that eventually the honest validators' n_v values become large enough for the lemma to apply. On the other hand, if the n_v are very large, new blocks will only be created very infrequently and the blockchain's progress will be slow; so it's important to optimize the choice of these parameters.

This is the motivation for the protocol's name: The nodes have different speed parameters n_v, as if they were driving on different lanes of a highway.

3.2 Parameter strategy: Target Threshold for Finality

Given a family $\mathcal{F}$ of voting functions, a threshold $0\% < D < 100\%$, a *break parameter* $C \in \mathbb{N}$, and an *acceleration parameter* $B \in \mathbb{N}$, we choose n_v as follows. Given i, and $m = n_v(i-1)$, whenever 2^m divides i and n_v was constant between $i - C \cdot 2^m$ and $i - 1$:

- If $i/2^m$ is divisible by 2, for no $k \in \{1, \dots, C\}$, there is an $f \in \mathcal{F}$ and a leader message λ in tick $i - k \cdot 2^m$, such that f was $\mathbb{0}$ for all messages in $J(\lambda)$, $f(\lambda) \neq \emptyset$, and $f(\lambda)$ was finalized with threshold D in our local protocol state by time $i - (k-1)2^m$, then let $n_v(i) = m + 1$.
- Otherwise, if $i/2^m$ is divisible by B, let $n_v(i) = m - 1$.

In other words, if C consecutive leaders have failed to finalize a new value with threshold D, then we increase n_v; but every B rounds, we optimistically decrease it.

In practice D should probably be about 1%: Even if in a single round, the leader's value gets finalized with only that threshold, if there are enough honest validators, its threshold will increase in the next round anyway. C should be chosen so that it is rare for C faulty leaders to occur in a row (resulting in an unnecessary slowdown). In particular, during B rounds, the expected number of such a faulty streak must be much less than 1, e.g. $C = 15$, $B = 1000$.

Proposition 2. *If there is a set H of honest nodes of weight at least $(1/2 + D)\mathbb{w}(\mathcal{V})$, all leaders in H set a first non-$\mathbb{0}$ value for some $f \in \mathcal{F}$, and there is some upper bound such that all messages between members of H are eventually delivered within that bound, then eventually some value gets finalized with threshold D.*

Proof. Every C rounds where nothing gets finalized, each honest validator v increments their n_v. Eventually, all members of H will have slow enough rounds to satisfy the requirements of Lemma 2. □

3.3 Leader Selection and Weight Readjustment

The leader-based algorithm requires all nodes to agree on the same leader schedule, which creates some difficulties. To begin with, it makes sense to assign leaders pseudorandomly, with probability proportional to weights in order to guarantee that honest leaders are selected a reasonable fraction of the time (as dishonest leaders can disrupt liveness or leave out transactions). Unfortunately, this creates a chicken-and-egg problem since the leader schedule in a network with dynamic weights will depend on the contents of some earlier block. Another issue is that an attacker might try to modify their stakes in a way that causes the known pseudorandom number generator to assign leadership to the attacker's nodes many times in a row.

To help resolve this, we will keep the validator set and their weights constant for long periods of times, e.g. for one week. We call such a period an *era*. To address the above issues, we need to ensure that:

- The block deciding the next era's sequence of leaders, i.e. the seed for the PRNG and the set and weights of the validators, is finalized with a very high fault tolerance by the end of an era.
- The validator creating that block does not have an opportunity to choose among a large number of leader sequences.

The first point means that for every era there will be a *key block* that gets created and finalized a long time (e.g. five days) before the beginning of the era, which fully determines the sequence of leaders. Later blocks cannot affect that sequence anymore.

The second point means that the creator of the key block must already be limited in their choice. We will make it so that they can only choose one out of two possible leader sequences.

So the full protocol requires several additional parameters:

- The length t_e of an era.
- The delay t_k between a key block and its era.
- The delay $t_v > t_k$ between the first block that contributes to the determination of the new leader sequence, and its era.
- The time t_f after the end of an era that the previous validators continue voting.

Each block whose parent has a timestamp between kt_e and $(k+1)t_e$ belongs to era number k. Every block with a timestamp between $kt_e - t_v$ and $kt_e - t_k$ contains a random bit chosen by the block's sender.

The first block with a timestamp $\geq kt_e - t_k$ is the *key block* for era k, and the first block with a timestamp $\geq kt_e - t_v$ is the *booking block*.

The validator weights $\mathbb{w}_b$ for a block in era k are the ones implied by the on-chain governance smart contract state in the unique booking block that is an ancestor of b. (See section 4.1 for details.)

And the leadership schedule a validator with state σ uses during era k, i.e. between time kt_e and $(k+1)t_e$, assigns to each tick i the outcome of a PRNG selecting a validator with weighted probabilities $\mathbb{w}_b$ seeded with:

- the hash of b and
- the sequence of random bits of the blocks between b and the unique key block that is an ancestor of c,

where c is the GHOST choice of σ and b is the unique booking block that is an ancestor c.

Between $(k+1)t_e$ and $(k+1)t_e + t_f$, validators from era k still participate in the protocol, even though they can't be leaders again and are only allowed to send ballots (except if they remain active in era $k+1$ as well). This is to ensure that the last blocks of era k still get finalized, even if many of the era's validators leave in era $k+1$.

This all works well if the key block is finalized with a sufficient threshold by the time a new era begins. If that is not the case, however, the network's liveness is in danger. Therefore t_k should be large enough to not only finalize the key block, but also for checkpoints to be exchanged and manually verified between the key block and the start of the era. (See section 4.2.)

4 A Permissionless Network

4.1 Proof of Stake

So far we proved the desirable properties of the protocol under the assumption that a certain fraction of validators (by weight) is well-behaved. A real-world network's security can approximately be measured in the cost of an attack: If I wanted to bribe the validators into reverting a finalized transaction or stalling the network, how much would I have to pay?

In a proof of stake network, these aspects are connected by making the weight proportional to a deposit — the validator's *stake* — that is locked for a period beginning sufficiently long before the weight was assigned, and ending sufficiently long after it was unassigned. Incorrect behavior, as far as it can be determined objectively, is punished automatically by removing some amount from the deposit. Making stakes is incentivized by rewarding correct behavior.

All of this can be implemented as a smart contract running inside the system itself, with two special hooks:

- The state of the smart contract determined by each block b (i.e. the result of executing the

transactions in all of b's ancestors including b itself) defines the weight function $\mathbb{w}_b$ after that block.

- The system calls the smart contract in each block even if no user made a transaction interacting with it. In that call, the system provides the contract with information about validators' behavior.

In addition, users can make transactions that call the contract to deposit or withdraw stakes.

The contract determines the weights for era k using only the deposits that have been made before $kt_e - t_\mathcal{V}$. Withdrawal requests must only be executed with a much longer delay: Between the era's end and the payout, checkpoints must have been finalized and exchanged so that the validator has no power anymore to revert any transactions. (See section 4.2.)

The exact conditions and percentages for penalties are beyond the scope of this document; they depend on external economic factors and the exact use case. In general, though:

- Creating an equivocation should cause a large percentage of the stake to be confiscated, possibly 100%: Equivocations can cause finalized transactions to be reverted, and they can only be caused by malice or severe software bugs, so they need to be extremely expensive.
- Liveness faults are less clear: Nodes can be offline because of network or power outages; which is indistinguishable from an attack on the network's liveness. In general, guaranteeing a certain level of reliability must be part of the validator's task and what they vouch for with their stakes. But, unrealistic requirements discourage participation in the network. One compromise is to measure the time in which a validator has produced no messages, and superlinearly, increase the percentage that is confiscated.
- In addition, nodes should include their values n_v in their messages, as a commitment to follow the schedule with the corresponding rhythm. From that it can be computed in which slots they are expected to produce blocks, and at what points in time they must unconditionally create ballots. (See section 3.)

Liveness faults are indistinguishable from censorship: If a majority of validators refuses to include a minority's messages, from the protocol's perspective this looks exactly like that minority failing to send messages. To discourage censorship, it might be necessary to incur a penalty not only on the validators seemingly exhibiting the liveness fault, but also on all the other validators. The precise design of such an incentive mechanism is beyond the scope of this paper.

4.2 Checkpoints

Like all proof of stake protocols, ours is vulnerable to long range attacks: For every past era k whose key block was finalized with a threshold of x, it is still possible for more than x of its validators to collude and to rewrite history starting from that point. That means there is a

growing list of past sets of validators that the network's security relies on, each of which have the opportunity to revert thousands of transactions. If the network has grown in value since then, their stakes might have been much lower than the network's current total stakes; and if they have already ceased to be validators and withdrawn and sold their stakes, they have nothing to lose anymore.

This can be addressed with out-of-protocol checkpoints, similar to [But14]:

A checkpoint is a block that the user can configure to be irreversibly finalized: The node will refuse to accept any blocks that are not descendants or ancestors of every checkpoint. It will act as if a validator with infinite weight had voted for all checkpoints.

Automatically turning every sufficiently finalized block into a checkpoint is an effective defense against long range attacks. However, that comes at a cost: What happens when the block in question *does* get orphaned at the precise point in time where we have turned it into a checkpoint, but most other nodes have *not* yet done so?

In truth, the checkpoint functionality just shifts the consensus problem to another timescale: one of weeks or months instead of seconds or minutes. On that timescale, consensus can happen semi-automatically instead of fully automatically:

Nodes should indeed automatically turn blocks into checkpoints, but in addition, they will implement functionality to publish those checkpoints and to receive them from different sources e.g.:

- One day after the end of an era, automatically make the last block of that era (according to the local GHOST rule) a checkpoint.
- Automatically publish your checkpoint via different channels: websites, blogs, social media.
- Subscribe to your friends checkpoint channels, and to those of some media you trust.
- Whenever there are any two checkpoints that are not ancestors of each other, alert the user: An exceptional fork has happened, or one of our trusted checkpoint sources has gone bad. At this point, the user needs to remove the untrusted checkpoints and subscriptions, and manually choose a fork.

References

[Bai16] Leemon Baird. "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance". In: *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.* (2016). URL: https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf.

[But14] Vitalik Buterin. "Proof of stake: How I learned to love weak subjectivity". In: *Ethereum blog* 25 (2014). URL: https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/.

[C+99] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186. URL: http://pmg.csail.mit.edu/papers/osdi99.pdf.

[Mil+16] Andrew Miller et al. "The honey badger of BFT protocols". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 31–42. URL: https://eprint.iacr.org/2016/199.pdf.

[Zam+18] V Zamfir et al. "Introducing the minimal CBC Casper family of consensus protocols". In: *DRAFT v1. 0* 5 (2018). URL: https://github.com/cbc-casper/cbc-casper-paper/blob/master/cbc-casper-paper-draft.pdf.