Introducing the "Minimal CBC Casper" Family of Consensus Protocols

DRAFT v1.0

Vlad Zamfir, Nate Rush, Aditya Asgaonkar, Georgios Piliouras

Ethereum Research

November 5, 2018

Abstract

CBC Casper is a family of "correct-by-construction" consensus protocols that share the same proof of asynchronous, Byzantine fault tolerant consensus safety. We describe this family of protocols by defining their protocol states and protocol state transitions, and then we provide a proof of Byzantine fault tolerant consensus safety for the entire Minimal CBC Casper family of protocols. We give examples members of this family of protocols, including a binary consensus protocol. "Casper the Friendly Ghost", a blockchain consensus protocol, and a sharded blockchain consensus protocol. Each of these examples is "correct-by-construction" because the way they are defined guarantees that they are part of this family of protocols, and therefore satisfy the consensus safety theorem. This draft is intended to be introductory and educational, it is not a complete description of a system that can be implemented exactly as specified.

Contents

1 Introduction

1.1 Consensus Protocols

Consensus protocols are used by nodes in a distributed network to make consistent decisions out of possibly inconsistent alternative decisions. For example, in a binary consensus protocol a decision on "1" is said to be inconsistent with a decision on "0" and consistent with a decision on "1". In a blockchain consensus protocol, a decision on one block is called "inconsistent" with another when they are not in the same blockchain, and "consistent" when they are in the same chain.

More specifically, consensus protocols traditionally are said to provide two properties:

- **Safety**: It's not possible for nodes to make inconsistent decisions
- **Liveness**: Nodes eventually/inevitably make decisions

A consensus protocol is said to be "synchronously safe" (or "live in a synchronous network"), if its safety (or liveness) is guaranteed by an assumption about timing, or about the way that race conditions get resolved (e.g. all these messages will arrive before some timeout). It is called "asynchronously safe" (or "live in asynchrony") if its safety (or liveness) is guaranteed without assumptions about timing, or about the way that race conditions get resolved (with the exception that every message sent is eventually recieved).

Fault tolerant consensus protocols have safety and/or liveness despite some number of *faults*. Traditionally, faulty behaviour is either considered to be "Crash Faulty", where nodes are faulty by virtue of not sending messages, or they are considered "Byzantine", which means that they may exhibit arbitrary behaviour. There is famous result known as "FLP impossibility" [1] that shows that it's impossible to have a deterministic consensus protocol that is safe and live in an asynchronous network, if there is one crash fault.

More specifically, we use the following fault classification:

Liveness faults:

- Crash fault: failing to send a message when expected
- Ommission fault: failing to receive a message when expected

Safety faults:

- Invalid message fault: producing messages that cannot be produced by a protocol following node
- Equivocation fault: producing valid messages in a way that could not have been produced by a *single* execution of the protocol

They are named this way because liveness faults are indistinguishable from network latency (from any point in the distributed system that isn't the faulty node itself), and because network latency (and therefore liveness faults) cannot cause protocol-following nodes to make inconsistent decisions in an asynchronously safe consensus protocol. We assumed that nodes can check the validity of messages, so if we insist that they do, then only equivocation faults will be able to cause consensus failure.

In a non-trivial consensus protocol, protocol-following nodes following completely independent executions of the protocol can decide on inconsistent values. Equivocating nodes can (by definition) exclusively show different nodes messages from independent protocol executions, and as a result, a large enough number of equivocation faults is fundamentally enough to cause consensus failure in any consensus protocol.

1.2 About This Paper

This document specifies a family of asynchronously safe consensus protocols in terms of the protocol states (sets of messages) and protocol state transitions (the superset relation) that clients running the protocol will execute

(by receiving messages from consensus forming nodes). Each member of the family satisfies a proof (shared here) that if there aren't more than some number (weight) of equivocations then it is impossible for nodes to make inconsistent decisions. This number, the protocol's equivocation fault tolerance threshold, is given as a parameter of the CBC Casper family of protocols. Equivocation faults are a subset of Byzantine faults and we have consensus safety unless there are too many equivocation faults specifically, meaning that the consensus protocols tolerate the equivocation fault tolerance threshold number (weight) of Byzantine faults.

Each member of the family is going to be identified by five "parameters". Four of these parameters can be given before the protocol specification, but one ("the estimator") must initially be left undefined because it is a function of the (initially undefined) protocol states. The specification of the family of protocols is a partial specification of any member of the family, because the parameters need to be defined in order for a consensus protocol to be specified. It is possible to choose the parameters in a way that corresponds to a trivial consensus protocol, so we will give some non-trivial example protocols by giving example parameters.

Note that this work does not make any synchrony assumptions, nor does it discuss liveness. It doesn't say anything about when validators send or receive messages. It doesn't address validator rotation, incentive mechanisms, proof-of-stake, light clients, scalability, or message source authentication. The consensus protocols specified here therefore should not be considered completely specified even after the parameters are specified.

1.3 Prerequisite Notation

Notation	**Description**
$\neg p,\ p \wedge q,\ p \vee q$	boolean negation, "and", "or"
$p \implies q,\ p \iff q$	boolean implication and equivalence
$X = \{x_1, x_2, \ldots, x_n\}$	set notation
$\{\} = \emptyset$	the empty set
$X \cup Y,\ X \cap Y,\ X \setminus Y$	set union, intersection, difference
$X \subseteq Y, X \subset Y$	improper subset and proper subset
$X \times Y$	Cartesian product
$x \in X$	set membership
$\forall x \in X,\ \exists y \in Y$	universal and existential quantifiers
Y^X	the set of maps from X to Y
$\{y \in Y : \exists x \in X, f(x) = y\}$	set builder notation
$\mathbb{N},\ \mathbb{Z},\ \mathbb{R}$	Natural numbers, integers, real numbers
$\sum_{x \in X} W(x),$	"Sigma" sum notation
$\lambda x.f(x, y)$	lambda (anonymous) function creation

1.4 Acknowledgements

We'd like to thank Leslie Lamport and other pre-blockchain consensus protocol researchers for founding the field that left us with excellent definitions and some interesting consensus protocols, Satoshi Nakamoto for Nakamoto ("Proof-of-Work") consensus[2], as it served as an excellent introduction to Byzantine fault tolerant consensus and because its "forking" behaviour directly led to this work, and Greg Meredith for work on the definition of the protocol message data structure and for the introduction to the "correct-by-construction" methodology. We would finially like to thank Vitalik Buterin for his continued support, collaboration, and mentorship since the very early stages of this research, over three years ago.

2 Description of CBC Casper

2.1 CBC Casper "Parameters"

Our first parameter is a set of names that will label the senders of protocol messages. The senders are understood to be the consensus forming nodes, called "validators":

Definition 2.1 (Validator names)**.** A non-empty set:

$$\mathcal{V}$$

Without loss of generality (and in preparation for proof-of-stake), we assign a weight to each validator:

Definition 2.2 (Validator weights)**.** A function:

$$\mathcal{W} : \mathcal{V} \to \mathbb{R}_+$$

The CBC Casper protocol states are parametric in a Byzantine fault tolerance threshold:

Definition 2.3 (Byzantine fault tolerance threshold)**.** A non-negative real number, strictly smaller than the total validator weights

$$0 \leq t < \sum_{v \in \mathcal{V}} \mathcal{W}(v)$$

The protocols will be guaranteed to have consensus safety tolerating up to "t Byzantine faults", *measured by weight*.

Different protocols in the CBC Casper family can make decisions regarding different values:

Definition 2.4 (Consensus values)**.** A multi-element set:

$$\mathcal{C}$$

In a binary consensus protocol, $\mathcal{C} = \{0, 1\}$, while in a blockchain consensus protocol, $\mathcal{C}$ is the set of all possible blockchains.

The final parameter, called "the estimator", relates CBC Casper protocol states to sets of consensus values (subsets of $\mathcal{C}$). To give the function signature of this parameter we need to define the CBC Casper protocol states, Σ. However, we will leave this for the protocol definition section, and for now simply write:

Definition 2.5 (Estimator)**.** A function

$$\mathcal{E} : \Sigma \to \mathcal{P}(\mathcal{C}) \setminus \emptyset$$

Where $\mathcal{P}$ is the powerset function. When the estimator returns a multi-element set, validators will have an opportunity to choose non-deterministically between consensus values. Because domain of the estimator has not yet been defined, it isn't quite proper for us ask for the estimator a parameter. However, for ease of presentation we will assume that we have the estimator as a parameter.

Minimal CBC Casper protocol states will therefore be determined by the choice of these parameters:

- Validator names $\mathcal{V}$,
- Validator weights $\mathcal{W}$,
- Fault tolerance threshold t,
- Consensus values $\mathcal{C}$, and
- Estimator $\mathcal{E}$

2.2 Protocol Definition

We will define protocol states (Σ) as *certain* sets of messages, and messages (M) as *certain* triples of the form (consensus value, validator name, protocol state). Our target is therefore something that satisfies these formulas:

$$\Sigma \subset \mathcal{P}(M) \tag{1}$$
$$M \subset \mathcal{C} \times \mathcal{V} \times \Sigma \tag{2}$$

We will call the consensus value the "estimate" of a message, the validator name the "sender", and the protocol state the "justification", and define these corresponding convenience functions:

Definition 2.6 (Estimate, Sender, Justification)**.**

$$Estimate : M \to \mathcal{C} \tag{3}$$
$$Sender : M \to \mathcal{V} \tag{4}$$
$$Justification : M \to \Sigma \tag{5}$$
$$Estimate((c, v, \sigma)) := c \tag{6}$$
$$Sender((c, v, \sigma)) := v \tag{7}$$
$$Justification((c, v, \sigma)) := \sigma \tag{8}$$

We can construct finite protocol states that satisfy these equations by using the empty set of messages as the base case:

Definition 2.7 (Preliminary protocol states, Σ, protocol messages M)**.**

$$\Sigma^0 := \{\emptyset\} \tag{9}$$
$$M^n := \{m \in \mathcal{C} \times \mathcal{V} \times \Sigma^n : Estimate(m) \in \mathcal{E}(\sigma)\} \tag{10}$$
$$\Sigma^n := \{\sigma \in \mathcal{P}(M^{n-1}) : m \in \sigma \implies Justification(m) \subseteq \sigma\} \quad \text{for } n > 0 \tag{11}$$
$$M := \bigcup_{i=0}^{\infty} M^i \tag{12}$$
$$\Sigma := \bigcup_{i=0}^{\infty} \Sigma^i \tag{13}$$

The type signature of the estimator is $\mathcal{E} : \Sigma \to \mathcal{P}(\mathcal{C}) \setminus \emptyset$, causing possible concern due to our application of $\mathcal{E}$ to $Justification(m) \in \Sigma^n \neq \Sigma$. However we note that $\Sigma^n \subset \Sigma$, meaning that $\mathcal{E}$ is defined over Σ^n.

Note that we explicitly restricted protocol messages to only have consensus values from the estimator, allowing validators to make a choice between values of the consensus when the the estimator returns a multi-element set (but not otherwise).

We also restricted the definition of protocol states so that we could have the following definition of protocol state transitions:

Definition 2.8 (Protocol state transitions $\to$)**.**

$$\to : \Sigma \times \Sigma \to \{True, False\} \tag{14}$$
$$\sigma_1 \to \sigma_2 :\Leftrightarrow \ \sigma_1 \subseteq \sigma_2 \tag{15}$$

We will write this as $\sigma \to \sigma'$ and we will write $\sigma \nrightarrow \sigma'$ when there isn't a state transition from σ to σ'.

We now pivot towards defining the protocol states for the CBC Casper protocol, with fault tolerance threshold t, $\Sigma_t \subset \Sigma$. But first, we need to talk about equivocation (a kind of Byzantine fault).

Equivocation faults are a Byzantine behaviour that is fundamentally enough to cause consensus failure in any consensus protocol, and additionally, CBC Casper protocols are guaranteed to be safe as long as not more than t weight of validators are equivocating (and therefore Byzantine).

Definition 2.9 (Equivocating messages)**.**

$$\cdot \perp \cdot : M \times M \to \{True, False\} \tag{16}$$
$$m_1 \perp m_2 :\Leftrightarrow \ Sender(m_1) = Sender(m_2) \wedge m_1 \neq m_2 \tag{17}$$
$$\wedge\, m_1 \notin Justification(m_2) \wedge m_2 \notin Justification(m1) \tag{18}$$

Equivocating messages are distinct messages from the same validator that do not include each other in their justifications.

The equivocating validators in a protocol state are simply the validators who have equivocating messages in that state:

Definition 2.10 (Equivocating validators)**.**

$$E : \Sigma \to \mathcal{P}(\mathcal{V}) \tag{19}$$
$$E(\sigma) := \{v \in \mathcal{V} : \exists m_1 \in \sigma, \exists m_2 \in \sigma,\ m_1 \perp m_2\ \wedge\ Sender(m_1) = v\} \tag{20}$$

Equivocation corresponds to behaviour that isn't "single threaded". We will eventually see a lemma showing that non-equivocating validators have at most one latest message.

Instead of considering the number of equivocations, we measure equivocations by weight:

Definition 2.11 (Equivocation fault weight)**.**

$$F : \Sigma \to \mathbb{R}_+ \tag{21}$$
$$F(\sigma) := \sum_{v \in E(\sigma)} \mathcal{W}(v) \tag{22}$$

Note that F is a monotonic function (i.e. $\sigma_1 \subseteq \sigma_2 \implies F(\sigma_1) \leq F(\sigma_2)$).

The CBC Casper protocol states will be parametric in an equivocation fault tolerance threshold t, and will use the following protocol states:

Definition 2.12 (Protocol states (with equivocation fault tolerance t))**.**

$$\Sigma_t = \{\sigma \in \Sigma : F(\sigma) \leq t\}$$

We think of protocol-following nodes as existing in these protocol states (Σ_t), and following the protocol state transitions ($\to$) to get from one state to another. If a node at state $\sigma \in \Sigma_t$ receives messages m such that $\sigma \cup \{m\} \in \Sigma$, then they will transition to state $\sigma \cup \{m\}$, if it doesn't expose the node to too many equivocation faults (i.e., as long as $F(\sigma \cup \{m\}) \leq t$).

We are now finished the protocol definition and are ready to prove consensus safety for the family of protocols that satisfy this definition (in the context of less than t equivocations (by weight)).

3 Safety Proof

Our obligation is to provide a way for nodes to make consistent decisions even if they receive different messages, as long as there are less than t equivocation faults in their protocol states. We do this in two steps, first by guaranteeing that nodes will have common future protocol states (as long as there are less than t equivocation faults *in the union of their protocol states*), and then by showing that their decisions on properties of protocol states will be consistent (if they share common future states, which they will if there aren't too many faults). Finally, we will show how this result can be leveraged to guarantee the consistency of decisions on properties of consensus values.

3.1 Guaranteeing Common Futures

We will define the common futures of a protocol states in terms of the intersection of their "futures cones":

Definition 3.1 (Futures cone in Σ_t)**.**

$$Futures_t : \Sigma_t \to \mathcal{P}(\Sigma_t) \tag{23}$$

$$Futures_t(\sigma) := \{\sigma' \in \Sigma_t : \sigma \to \sigma'\} \tag{24}$$

Note that the futures function is monotonic in protocol state transitions, meaning that the futures cone "shrinks" during protocol execution:

Lemma 1 (Monotonic futures)**.** $\forall \sigma \in \Sigma_t$, $\forall \sigma' \in \Sigma_t$,

$$\sigma' \in Futures_t(\sigma) \iff Futures_t(\sigma') \subseteq Futures_t(\sigma)$$

Proof. (Forwards direction)

$$\begin{aligned}
\sigma' \in Futures_t(\sigma) &\iff \sigma \to \sigma' && (25)\\
&\iff \sigma \to \sigma' \wedge \forall \sigma''' \in Futures_t(\sigma'), \sigma' \to \sigma''' && (26)\\
&\iff \sigma \to \sigma' \wedge \forall \sigma''' \in Futures_t(\sigma'), \sigma' \to \sigma''' \wedge \sigma \to \sigma''' && (27)\\
&\implies \forall \sigma''' \in Futures_t(\sigma'), \sigma' \to \sigma''' \wedge \sigma \to \sigma''' && (28)\\
&\implies \forall \sigma''' \in Futures_t(\sigma'), \sigma \to \sigma''' && (29)\\
&\iff \forall \sigma''' \in Futures_t(\sigma'), \sigma''' \in Futures_t(\sigma) && (30)\\
&\iff Futures_t(\sigma') \subseteq Futures_t(\sigma) && (31)
\end{aligned}$$

(Backwards direction)

$$\begin{aligned}
Futures_t(\sigma') \subseteq Futures_t(\sigma) &\iff \forall \sigma'' \in Futures_t(\sigma'), \sigma'' \in Futures_t(\sigma) && (32)\\
&\iff \forall \sigma'' \in Futures_t(\sigma'), \sigma'' \in Futures_t(\sigma) \wedge \sigma' \in Futures_t(\sigma') && (33)\\
&\implies \sigma' \in Futures_t(\sigma) && (34)
\end{aligned}$$

■

We will now present our first key theorem, which guarantees that pairs of nodes have common future states if not more than t fault weight is observed in the union of their states:

Theorem 1 (2-party common futures)**.** $\forall \sigma_1 \in \Sigma_t$, $\forall \sigma_2 \in \Sigma_t$,

$$F(\sigma_1 \cup \sigma_2) \leq t \implies Futures_t(\sigma_1) \cap Futures_t(\sigma_2) \neq \emptyset$$

Proof.

$$\begin{aligned}
F(\sigma_1 \cup \sigma_2) \leq t &\iff \sigma_1 \cup \sigma_2 \in \Sigma_t && (35)\\
&\iff \sigma_1 \cup \sigma_2 \in \Sigma_t \wedge \sigma_1 \to \sigma_1 \cup \sigma_2 \wedge \sigma_2 \to \sigma_1 \cup \sigma_2 && (36)\\
&\iff \sigma_1 \cup \sigma_2 \in \Sigma_t \wedge \sigma_1 \to \sigma_1 \cup \sigma_2 \wedge \sigma_1 \cup \sigma_2 \in \Sigma_t \wedge \sigma_2 \to \sigma_1 \cup \sigma_2 && (37)\\
&\iff \sigma_1 \cup \sigma_2 \in Futures_t(\sigma_1) \wedge \sigma_1 \cup \sigma_2 \in Futures_t(\sigma_2) && (38)\\
&\iff \sigma_1 \cup \sigma_2 \in Futures_t(\sigma_1) \cap Futures_t(\sigma_2) && (39)\\
&\implies Futures_t(\sigma_1) \cap Futures_t(\sigma_2) \neq \emptyset && (40)\\
& && (41)
\end{aligned}$$

■

Note the analagous guarantee that n nodes have a common future if there aren't too many faults in the union of their states:

Theorem 2 (n-party common futures)**.** $\forall \sigma_i \in \Sigma_t$,

$$F(\bigcup_{i=1}^{n} \sigma_i) \leq t \implies \bigcap_{i=1}^{n} Futures_t(\sigma_i) \neq \emptyset$$

Proof. Similar to proof of the 2-argument version. ■

The theorems above guarantee that protocol-following nodes have a common future state if the validators that are equivocating in the union of their states have no more than t weight. Therefore, if Byzantine (*and equivocating*) validators have no more than t weight, all protocol-following nodes will have a common future state.

We will use this result to ensure the consistency of decisions made by protocol-following nodes, guaranteeing the consensus safety of CBC Casper consensus protocols.

3.2 Guaranteeing Consistent Decisions

In this section, we will show that if nodes have a common future state, then all of the invariant properties of their protocol states are consistent. We will guarantee consensus safety by insisting that nodes have decided on the invariant properties of their protocol states. We will then leverage the consistency of decisions on properties of protocol states to guarantee the consistency of decisions on properties of consensus values.

3.2.1 Guaranteeing Consistent Decisions on Properties of Protocol States

Before we can formally state our consensus safety theorems, we need to define some preliminary notions. First, we consider properties of protocol states to be maps from protocol states to $True$ or $False$:

Definition 3.2 (Properties of protocol states)**.**

$$P_\Sigma = \{f \in \{True, False\}^\Sigma\}$$

We say that property of a protocol state is called "decided" from some protocol state if it holds for all future protocol states:

Definition 3.3 (Decided properties of protocol states)**.**

$$Decided_{\Sigma,t} : P_\Sigma \times \Sigma \to \{True, False\} \tag{42}$$

$$Decided_{\Sigma,t}(p, \sigma) :\Leftrightarrow \forall \sigma' \in Futures_t(\sigma), p(\sigma') \tag{43}$$

We will now prepare lemmas that we will use to prove consensus safety.

Our first lemma says that if a node is decided on a property at some state, then it will be decided on this property at its future protocol states:

Lemma 2 (Forward consistency)**.** $\forall \sigma \in \Sigma_t$, $\forall \sigma' \in \Sigma_t$, $\forall p \in P_\Sigma$,

$$\sigma' \in Futures_t(\sigma) \implies (Decided_{\Sigma,t}(p, \sigma) \implies Decided_{\Sigma,t}(p, \sigma'))$$

Proof.

$$\sigma' \in Futures_t(\sigma) \wedge Decided_{\Sigma,t}(p, \sigma) \tag{44}$$

$$\iff Futures_t(\sigma') \subseteq Futures_t(\sigma) \wedge Decided_{\Sigma,t}(p, \sigma) \tag{45}$$

$$\iff Futures_t(\sigma') \subseteq Futures_t(\sigma) \wedge \forall \sigma'' \in Futures_t(\sigma), p(\sigma'') \tag{46}$$

$$\implies \forall \sigma'' \in Futures_t(\sigma'), p(\sigma'') \tag{47}$$

$$\iff Decided_{\Sigma,t}(p, \sigma') \tag{48}$$

So now we have

$$\sigma' \in Futures_t(\sigma) \land Decided_{\Sigma,t}(p,\sigma) \implies Decided_{\Sigma,t}(p,\sigma') \tag{49}$$
$$\implies \sigma' \in Futures_t(\sigma) \implies (Decided_{\Sigma,t}(p,\sigma) \implies Decided_{\Sigma,t}(p,\sigma')) \tag{50}$$
$$\tag{51}$$

■

Our second lemma says that if a node has a decided property at some state, then the negation that property must not have been decided at any possible past state.

Lemma 3 (Backwards consistency)**.** $\forall \sigma \in \Sigma_t, \forall \sigma' \in \Sigma_t, \forall p \in P_\Sigma$,

$$\sigma' \in Futures_t(\sigma) \implies (Decided_{\Sigma,t}(p,\sigma') \implies \neg Decided_{\Sigma,t}(\neg p,\sigma))$$

Where $\neg$ denotes negation and is defined as expected.

Proof.

$$\sigma' \in Futures_t(\sigma) \land Decided_{\Sigma,t}(p,\sigma') \tag{52}$$
$$\iff Futures_t(\sigma') \subseteq Futures_t(\sigma) \land Decided_{\Sigma,t}(p,\sigma') \tag{53}$$
$$\iff Futures_t(\sigma') \subseteq Futures_t(\sigma) \land \forall \sigma'' \in Futures_t(\sigma'), p(\sigma'') \tag{54}$$
$$\implies Futures_t(\sigma') \subseteq Futures_t(\sigma) \land \exists \sigma'' \in Futures_t(\sigma), p(\sigma'') \tag{55}$$
$$\implies \exists \sigma'' \in Futures_t(\sigma), p(\sigma'') \tag{56}$$
$$\iff \neg \forall \sigma'' \in Futures_t(\sigma), \neg p(\sigma'') \tag{57}$$
$$\iff \neg Decided_{\Sigma,t}(\neg p,\sigma) \tag{58}$$

So now we have

$$\sigma' \in Futures_t(\sigma) \land Decided_{\Sigma,t}(p,\sigma') \implies \neg Decided_{\Sigma,t}(\neg p,\sigma) \tag{59}$$
$$\implies \sigma' \in Futures_t(\sigma) \implies (Decided_{\Sigma,t}(p,\sigma') \implies \neg Decided_{\Sigma,t}(\neg p,\sigma)) \tag{60}$$
$$\tag{61}$$

■

We will now use these two lemmas to prove two-party consensus safety. We show this result before showing n-party consensus safety as it is considerably more accessible. Because nodes only decide on properties of protocol states, the only way two nodes can make inconsistent decisions is if one node is decided on a property p, and the other is decided on its negation $\neg p$. Our two-party consensus safety theorem therefore states that two nodes don't decide on p and $\neg p$ if there aren't more than t faults:

Theorem 3 (Two-party consensus safety)**.** $\forall \sigma_1 \in \Sigma_t, \forall \sigma_2 \in \Sigma_t, \forall p \in P_\Sigma$,

$$F(\sigma_1 \cup \sigma_2) \leq t \implies \neg(Decided_{\Sigma,t}(p,\sigma_1) \land Decided_{\Sigma,t}(\neg p,\sigma_2)) \tag{62}$$

Proof.

$$F(\sigma_1 \cup \sigma_2) \leq t \implies Futures_t(\sigma_1) \cap Futures_t(\sigma_2) \neq \emptyset \tag{63}$$
$$\iff \exists \sigma \in Futures_t(\sigma_1) \cap Futures_t(\sigma_2) \tag{64}$$
$$\iff \exists \sigma \in \Sigma_t, \sigma \in Futures_t(\sigma_1) \land \sigma \in Futures_t(\sigma_2) \tag{65}$$
$$\implies \exists \sigma \in \Sigma_t, (Decided_{\Sigma,t}(p,\sigma_1) \implies Decided_{\Sigma,t}(p,\sigma)) \land (Decided_{\Sigma,t}(p,\sigma) \implies \neg Decided_{\Sigma,t}(\neg p,\sigma_2)) \tag{66}$$
$$\implies Decided_{\Sigma,t}(p,\sigma_1) \implies \neg Decided_{\Sigma,t}(\neg p,\sigma_2) \tag{67}$$
$$\iff \neg Decided_{\Sigma,t}(p,\sigma_1) \lor \neg Decided_{\Sigma,t}(\neg p,\sigma_2) \tag{68}$$
$$\iff \neg(Decided_{\Sigma,t}(p,\sigma_1) \land Decided_{\Sigma,t}(\neg p,\sigma_2)) \tag{69}$$

■

Note the contrapositive of this theorem, which shows that the existence of nodes at two states with inconsistent decisions implies that there are more than t weight of validators equivocating:

$$\exists p \in P_\Sigma,\ Decided_{\Sigma,t}(p, \sigma_1) \wedge Decided_{\Sigma,t}(\neg p, \sigma_2) \implies F(\sigma_1 \cup \sigma_2) > t$$

Unfortunately, the two-party consensus safety result is not sufficient to guarantee the consistency of decisions made by more than 2 parties. To see this, consider a triple of properties p, q, r such that $p \wedge q \implies \neg r$. If three nodes decide on p, q and r respectively, then they will have made inconsistent decisions, although they may be consistent pairwise. We therefore need to understand inconsistent decisions in a more general way than we considered for the two-party consensus safety result, where looking at properties and their negations was sufficient.

We will say that properties are "inconsistent" if, for all protocol states, their conjunction is false:

Definition 3.4 (Inconsistency of properties of protocol states)**.**

$$Inconsistent_\Sigma : \mathcal{P}(P_\Sigma) \to \{True, False\}$$

$$Inconsistent_\Sigma(Q) :\Leftrightarrow \forall \sigma \in \Sigma, \bigwedge_{q \in Q} q(\sigma) = False$$

Then we will say that properties are "consistent" if they are not inconsistent:

$$\neg Inconsistent_\Sigma(Q) \iff \neg(\forall \sigma \in \Sigma, \bigwedge_{q \in Q} q(\sigma) = False) \tag{70}$$

$$\iff \exists \sigma \in \Sigma, \bigwedge_{q \in Q} q(\sigma) = True \tag{71}$$

$$\iff \exists \sigma \in \Sigma, \forall q \in Q, q(\sigma) \tag{72}$$

Which is to say that a set of properties is consistent if there is at least one protocol state that satisifies each of them:

Definition 3.5 (Consistency of properties of protocol states)**.**

$$Consistent_\Sigma : \mathcal{P}(P_\Sigma) \to \{True, False\}$$

$$Consistent_\Sigma(Q) :\Leftrightarrow \exists \sigma \in \Sigma, \forall q \in Q,\ q(\sigma)$$

The n-party consensus safety result will show the consistency of *all* the decided properties in *any* of the states of our n nodes, if there are not more than t equivocation faults by weight. So, we need to define a function that collects all the decisions from protocol states:

Definition 3.6 (Decisions on properties of protocol states)**.**

$$Decisions_{\Sigma,t} : \Sigma \to \mathcal{P}(P_\Sigma)$$

$$Decisions_{\Sigma,t}(\sigma) = \{p \in P_\Sigma : Decided_{\Sigma,t}(p, \sigma)\}.$$

Now, we state the n-party consensus safety result, that the decided properties are consistent (if there aren't too many faults):

Theorem 4 (n-party consensus safety for properties of protocol states)**.** $\forall \sigma_i \in \Sigma_t$,

$$F(\bigcup_{i=1}^{n} \sigma_i) \leq t \implies Consistent_\Sigma(\bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i))$$

Proof.

$$F(\bigcup_{i=1}^{n}\sigma_i) \leq t \implies \bigcap_{i=1}^{n} Futures_t(\sigma_i) \neq \emptyset \tag{73}$$

$$\iff \exists \sigma \in \Sigma_t, \sigma \in \bigcap_{i=1}^{n} Futures_t(\sigma_i) \tag{74}$$

$$\iff \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^{n} \sigma \in Futures_t(\sigma_i) \tag{75}$$

$$\iff \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^{n} \sigma \in Futures_t(\sigma_i) \tag{76}$$

$$\wedge \bigwedge_{j=1}^{n} \sigma \in Futures_t(\sigma_j) \implies (\forall p \in P_\Sigma, Decided_{\Sigma,t}(p, \sigma_j) \implies Decided_{\Sigma,t}(p, \sigma)) \tag{77}$$

$$\implies \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^{n} \forall p \in P_\Sigma, Decided_{\Sigma,t}(p, \sigma_i) \implies Decided_{\Sigma,t}(p, \sigma) \tag{78}$$

$$\implies \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^{n} \forall p \in Decisions_{\Sigma,t}(\sigma_i), Decided_{\Sigma,t}(p, \sigma_i) \implies Decided_{\Sigma,t}(p, \sigma) \tag{79}$$

$$\iff \exists \sigma \in \Sigma_t, \bigwedge_{i=1}^{n} \forall p \in Decisions_{\Sigma,t}(\sigma_i), Decided_{\Sigma,t}(p, \sigma) \tag{80}$$

$$\iff \exists \sigma \in \Sigma_t, \forall p \in \bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i), Decided_{\Sigma,t}(p, \sigma) \tag{81}$$

$$\implies \exists \sigma \in \Sigma_t, \forall p \in \bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i), p(\sigma) \tag{82}$$

$$\iff Consistent_\Sigma(\bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i)) \tag{83}$$

■

Note the contrapositive, that the existence of inconsistent decisions means that there must be more than t weight equiocating:

$$\neg Consistent_\Sigma(\bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i)) \implies F(\bigcup_{i=1}^{n}\sigma_i) > t$$

This concludes the consensus safety proof for decisions on properties of protocol states. We will now show how we can leverage this consensus safety theorem and the estimator to allow nodes to make consistent decisions on properties of consensus values.

3.2.2 Guaranteeing Consistent Decisions on Properties of Consensus Values

We will define properties of consensus values as the maps from consensus values to $True$ or $False$:

Definition 3.7 (Properties of consensus values)**.**

$$P_\mathcal{C} = \{f \in \{True, False\}^\mathcal{C}\}$$

For every property of consensus values there is a naturally corresponding property of protocol states, which is satisfied (by a protocol state) if the property of consensus holds for all values of the estimator (at that protocol state):

Definition 3.8 (H, naturally corresponding property of protocol states)**.**

$$H : P_\mathcal{C} \to P_\Sigma \tag{84}$$

$$H(p) :\Leftrightarrow \lambda\sigma.\forall c \in \mathcal{E}(\sigma), p(c) \tag{85}$$

For example, if we have an estimator for a binary consensus protocol, which has function signature $\mathcal{E} : \Sigma \to \mathcal{P}(\{0,1\}) \setminus \emptyset$, and the property of consensus values $p \in P_{\{0,1\}}$ with $p(0) = True$ and $p(1) = False$, then $H(p)$ is defined to be satisifed by a protocol state σ if $\mathcal{E}(\sigma) = \{0\}$, but not if $\mathcal{E}(\sigma) = \{1\}$ or $\mathcal{E}(\sigma) = \{0,1\}$.

We will define the consistency of properties of consensus values in the same way that we defined consistency of properties of protocol states, namely that they are consistent if there is a value of the consensus that satisifes all of them:

Definition 3.9 (Consistency of properties of the consensus)**.**

$$Consistent_{\mathcal{C}} : \mathcal{P}(P_{\mathcal{C}}) \to \{True, False\}$$

$$Consistent_{\mathcal{C}}(Q) :\Leftrightarrow \exists c \in \mathcal{C}, \forall q \in Q,\ q(c)$$

We will now show an important lemma, that if properties of protocol states that correspond to properties of consensus values are consistent, then these properties of consensus values are also consistent:

Lemma 4. $\forall p_i \in P_{\mathcal{C}}$,

$$Consistent_{\Sigma}(\{H(p_1), H(p_2), \ldots, H(p_n)\}) \implies Consistent_{\mathcal{C}}(\{p_1, p_2, \ldots, p_n\})$$

Proof.

$$\begin{aligned}
& Consistent_{\Sigma}(\{H(p_1), H(p_2), \ldots, H(p_n)\}) && (86)\\
\iff & \exists \sigma \in \Sigma, \forall i \in \{1,2,\ldots,n\}, H(p_i)(\sigma) && (87)\\
\iff & \exists \sigma \in \Sigma, \forall i \in \{1,2,\ldots,n\} \forall c \in \mathcal{E}(\sigma), p_i(c) && (88)\\
\iff & \exists \sigma \in \Sigma, \forall c \in \mathcal{E}(\sigma), \forall i \in \{1,2,\ldots,n\}, p_i(c) && (89)\\
\implies & \exists \sigma \in \Sigma, \exists c \in \mathcal{E}(\sigma), \forall i \in \{1,2,\ldots,n\}, p_i(c) && (90)\\
\iff & \exists c \in \mathcal{C}, \forall i \in \{1,2,\ldots,n\}, p_i(c) && (91)\\
\iff & Consistent_{\mathcal{C}}(\{p_1, p_2, \ldots, p_n\}) && (92)
\end{aligned}$$

■

We will use this lemma to ensure the consistency of properties of consensus values that correspond to decided properties of protocol states (via H). We will call these properties "the decided properties of consensus values":

Definition 3.10 (Decided on properties of consensus values)**.**

$$Decided_{\mathcal{C},t} : P_{\mathcal{C}} \times \Sigma_t \to \{True, False\} \quad (93)$$

$$Decided_{\mathcal{C},t}(p, \sigma) = Decided_{\Sigma,t}(H(p), \sigma) \quad (94)$$

In addition to deciding on properties of protocol states, nodes can also make decisions on properties of consensus values.

Definition 3.11 (Decisions on properties of consensus values)**.**

$$Decisions_{\mathcal{C},t} : \Sigma_t \to \mathcal{P}(P_{\mathcal{C}})$$

$$Decisions_{\mathcal{C},t}(\sigma) = \{q \in P_{\mathcal{C}} : H(q) \in Decisions_{\Sigma,t}(\sigma)\}$$

We are finally able to give our last consensus safety theorem, which shows that decisions on properties of consensus values are consistent if there aren't too many faults:

Theorem 5 (n-party consensus safety for properties of the consensus)**.** $\forall \sigma_i \in \Sigma_t$,

$$F(\bigcup_{i=1}^{n} \sigma_i) \leq t \implies Consistent_{\mathcal{C}}(\bigcup_{i=1}^{n} Decisions_{\mathcal{C},t}(\sigma_i))$$

Proof.

$$F(\bigcup_{i=1}^{n} \sigma_i) \leq t \tag{95}$$

$$\implies Consistent_{\Sigma}(\bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i)) \tag{96}$$

$$\implies Consistent_{\Sigma}(\{p \in \bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i) : \exists q \in P_{\mathcal{C}}, p = H(q)\}) \tag{97}$$

$$\iff Consistent_{\Sigma}(\bigcup_{q \in P_{\mathcal{C}} \ : \ H(q) \in \bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i)} \{H(q)\}) \tag{98}$$

$$\iff Consistent_{\mathcal{C}}(\{q \in P_{\mathcal{C}} : H(q) \in \bigcup_{i=1}^{n} Decisions_{\Sigma,t}(\sigma_i)\}) \tag{99}$$

$$\iff Consistent_{\mathcal{C}}(\bigcup_{i=1}^{n} Decisions_{\mathcal{C},t}(\sigma_i)) \tag{100}$$

■

And once again, we note the contrapositive of this theorem, which shows that if there are inconsistent decisions on properties of the consensus, then it must be that more than t weight of validators are Byzantine and equivocating:

$$\neg Consistent_{\mathcal{C}}(\bigcup_{i=1}^{n} Decisions_{\mathcal{C},t}(\sigma_i)) \implies F(\bigcup_{i=1}^{n} \sigma_i) > t$$

This concludes the safety proof for the minimal CBC Casper family of protocols.

4 Example Protocols

4.1 Preliminary Definitions

We need to develop a bit of language before giving example protocols. Specifically, we are going to define the terms required for us to talk about the estimates of the latest messages from non-equivocating validators.

The observed validators in a set of messages are all validators who have sent at least one of those messages:

Definition 4.1 (Observed validators)**.**

$$\begin{aligned} Observed :&\mathcal{P}(M) \to \mathcal{P}(\mathcal{V}) \\ Observed(\sigma) =&\{Sender(m) : m \in \sigma\} \end{aligned}$$

Definition 4.2.

$$\begin{aligned} Later :&M \times \mathcal{P}(M) \to \mathcal{P}(M) \\ Later(m, \sigma) =&\{m' \in \sigma : m \in Justification(m')\} \end{aligned}$$

The messages from a validator in a set of messages are all messages such that the sender of the message is that validator:

Definition 4.3 (Message From a Sender)**.**

$$\begin{aligned} From\_Sender :&\mathcal{V} \times \mathcal{P}(M) \to \mathcal{P}(M) \\ From\_Sender(v, \sigma) =&\{m \in \sigma : Sender(m) = v\} \end{aligned}$$

Similarly, we can define the messages from a group:

Definition 4.4 (Messages From a Group)**.**

$$\begin{aligned} From\_Group :&\mathcal{P}(\mathcal{V}) \times \mathcal{P}(M) \to \mathcal{P}(M) \\ From\_Group(V, \sigma) =&\{m \in \sigma : Sender(m) \in V\} \end{aligned}$$

Definition 4.5.

$$\begin{aligned} Later\_From :&M \times \mathcal{V} \times \mathcal{P}(M) \\ Later\_From(m, v, \sigma) =&Later(m, \sigma) \cap From\_Sender(v, \sigma) \end{aligned}$$

Definition 4.6 (Latest Message)**.**

$$\begin{aligned} L_M :&\mathcal{P}(M) \to (\mathcal{V} \to \mathcal{P}(M)) \\ L_M(\sigma)(v) =&\{m \in From\_Sender(v, \sigma) : Later\_From(m, v, \sigma) = \emptyset\} \end{aligned}$$

Definition 4.7 (Latest message driven estimator)**.**

$$\begin{aligned} Latest\_Message\_Driven :&\mathcal{P}(C)^{\Sigma} \to \{True, False\} \\ Latest\_Message\_Driven(\mathcal{E}) :&\Leftrightarrow \exists \hat{\mathcal{E}} \in \mathcal{P}(C)^{\mathcal{P}(M)^{\mathcal{V}}}, \quad \mathcal{E} = \hat{\mathcal{E}} \circ L_M \end{aligned}$$

Definition 4.8 (Latest Estimates)**.**

$$\begin{aligned} L_E :&\Sigma \to (\mathcal{V} \to \mathcal{P}(\mathcal{C})) \\ L_E(\sigma)(v) =&\{Estimate(m) : m \in L_M(\sigma)(v)\} \end{aligned}$$

Definition 4.9 (Latest message driven estimator)**.**

$$\begin{aligned} Latest\_Estimate\_Driven :&\mathcal{P}(C)^{\Sigma} \to \{True, False\} \\ Latest\_Estimate\_Driven(\mathcal{E}) :&\Leftrightarrow \exists \hat{\mathcal{E}} \in \mathcal{P}(C)^{\mathcal{P}(C)^{\mathcal{V}}}, \quad \mathcal{E} = \hat{\mathcal{E}} \circ L_E \end{aligned}$$

Lemma 5 (Non-equivocating validators have at most one latest message)**.** $\forall v \in \mathcal{V}$,

$$v \notin E(\sigma) \implies |L_M(\sigma)(v)| \leq 1$$

Proof. We will prove the contrapositive , $|L_M(\sigma)(v)| > 1 \implies v \in E(\sigma)$

$$\begin{aligned}
|L_M(\sigma)(v)| > 1 &\implies |\{m \in From_Sender(v,\sigma) : Later_From(m,v,\sigma) = \emptyset\}| > 1 \\
&\implies \exists m_1 \in From_Sender(v,\sigma), \exists m_2 \in From_Sender(v,\sigma), m_1 \neq m_2 \\
&\quad \wedge Later_From(m_1,v,\sigma) = \emptyset \wedge Later_From(m_2,v,\sigma) = \emptyset \\
&\iff \exists m_1 \in From_Sender(v,\sigma), \exists m_2 \in From_Sender(v,\sigma), m_1 \neq m_2 \\
&\quad \wedge Later(m_1,\sigma) \cap From_Sender(v,\sigma) = \emptyset \wedge Later(m_2,\sigma) \cap From_Sender(v,\sigma) = \emptyset \\
&\implies \exists m_1 \in From_Sender(v,\sigma), \exists m_2 \in From_Sender(v,\sigma), m_1 \neq m_2 \\
&\quad \wedge \nexists m^* \in From_Sender(v,\sigma), m^* \in Later(m_1,\sigma) \\
&\quad \wedge \nexists m^{**} \in From_Sender(v,\sigma), m^{**} \in Later(m_2,\sigma) \\
&\implies \exists m_1 \in From_Sender(v,\sigma), \exists m_2 \in From_Sender(v,\sigma), m_1 \neq m_2 \\
&\quad \wedge \nexists m^* \in From_Sender(v,\sigma), m_1 \in Justification(m^*) \\
&\quad \wedge \nexists m^{**} \in From_Sender(v,\sigma), m_2 \in Justification(m^{**}) \\
&\implies \exists m_1 \in From_Sender(v,\sigma), \exists m_2 \in From_Sender(v,\sigma), m_1 \neq m_2 \\
&\quad \wedge \forall m^* \in From_Sender(v,\sigma), m_1 \notin Justification(m^*) \\
&\quad \wedge \forall m^{**} \in From_Sender(v,\sigma), m_2 \notin Justification(m^{**}) \\
&\implies \exists m_1 \in \sigma : Sender(m_1) = v, \exists m_2 \in \sigma : Sender(m_2) = v, m_1 \neq m_2 \\
&\quad \wedge m_1 \notin Justification(m_2,\sigma) \wedge m_2 \notin Justification(m_1,\sigma) \\
&\implies \exists m_1 \in \sigma : Sender(m_1) = v, \exists m_2 \in \sigma : Sender(m_2) = v, \\
&\qquad Sender(m_1) = Sender(m_2) \wedge m_1 \neq m_2 \\
&\quad \wedge m_1 \notin Justification(m_2,\sigma) \wedge m_2 \notin Justification(m_1,\sigma) \\
&\implies \exists m_1 \in \sigma, \exists m_2 \in \sigma, Sender(m_1) = Sender(m_2) \wedge m_1 \neq m_2 \\
&\quad \wedge m_1 \notin Justification(m_2,\sigma) \wedge m_2 \notin Justification(m_1,\sigma) \\
&\iff v \in E(\sigma)
\end{aligned}$$

■

Definition 4.10 ($\preceq$)**.**

$$\begin{aligned}
\cdot \preceq \cdot : M \times M \to \{True, False\} \\
m_1 \preceq m_2 :\Leftrightarrow |Justification(m_1)| \geq |Justification(m_2))|
\end{aligned}$$

Lemma 6. $\forall \sigma \in \Sigma, \forall S \subseteq \sigma$

$(S, \preceq)$ is a total order.

Proof. Starting from the fact that all protocol states are finite, we show that every justification of every message

in σ (and in therefore in S) is finite.

$$\begin{aligned}
&\forall \sigma \in \Sigma, \exists n \in \mathbb{N}, n = |\sigma| \\
&\implies \forall \sigma \in \Sigma, \exists n \in \mathbb{N}, n = |\sigma| \\
&\quad \wedge \forall m \in \sigma, Justification(m) \subseteq \sigma \\
&\implies \forall \sigma \in \Sigma, \exists n \in \mathbb{N}, n = |\sigma| \\
&\quad \wedge \forall m \in \sigma, |Justification(m)| \leq |\sigma| \\
&\implies \forall \sigma \in \Sigma, \exists n \in \mathbb{N}, n = |\sigma| \\
&\quad \wedge \forall m \in \sigma, |Justification(m)| \leq n \\
&\implies \forall \sigma \in \Sigma, \exists n \in \mathbb{N}, n = |\sigma| \\
&\quad \wedge \forall m \in \sigma, \exists n' \in \mathbb{N}, n' = |Justification(m)| \\
&\implies \forall \sigma \in \Sigma, \\
&\quad \forall m \in \sigma, \exists n' \in \mathbb{N}, n' = |Justification(m)| \\
&\implies \forall \sigma \in \Sigma, \forall S \subseteq \sigma, \\
&\quad \forall m \in \sigma, \exists n' \in \mathbb{N}, n' = |Justification(m)| \\
&\implies \forall \sigma \in \Sigma, \forall S \subseteq \sigma, \\
&\quad \forall m \in S, \exists n' \in \mathbb{N}, n' = |Justification(m)|
\end{aligned}$$

This means that

$$\exists f \in \mathbb{N}^S, \forall m \in S, f(m) = |Justification(m)|$$

And we have the following equivalence for such a function f:

$$m_1 \preceq m_2 \iff |Justification(m_1)| \geq |Justification(m_2))| \iff f(m_1) \geq f(m_2)$$

Therefore, if $(Im(f), \geq)$ is a total order, then so is $(S, \preceq)$. Note that $Im(f)$ denoted the image of f, $\{n \in \mathbb{N} : \exists m \in S, f(m) = n\}$. Furthermore, we know that $(\mathbb{N}, \geq)$ is a total order, and it follows that $(Im(f), \geq)$ is a total order, which in turn means that $(S, \preceq)$ is a total order. ■

Lemma 7 (Monotonicity of Justifications)**.**

$$m' \in Later(m, \sigma) \implies Justification(m) \subseteq Justification(m')$$

Proof.

$$\begin{aligned}
&m' \in Later(m, \sigma) \\
\iff &m' \in \{m^* \in \sigma : m \in Justification(m^*)\} \\
\implies &m \in Justification(m') \\
\implies &\exists \sigma' \in \Sigma, m \in \sigma', \sigma' = Justification(m') \\
\implies &\exists \sigma' \in \Sigma, m \in \sigma', \sigma' = Justification(m') \wedge Justification(m) \subseteq \sigma' \\
\implies &\exists \sigma' \in \Sigma, m \in \sigma', Justification(m) \subseteq Justification(m') \\
\implies &Justification(m) \subseteq Justification(m')
\end{aligned}$$

■

Lemma 8. The minimal elements in $(From\_Sender(v, \sigma), \preceq)$ are the latest messages of validator v.

Let m be a minimal element in $(From\_Sender(v, \sigma), \preceq)$. Then m is a latest message iff $\{m \in From\_Sender(v, \sigma) : Later\_From(m, v, \sigma) = \emptyset\}$

Proof. (By Contradiction) Assume that a minimal element m is not a latest message.

$$
\begin{aligned}
&\forall m' \in From\_Sender(v,\sigma), m \preceq m' \land m \notin L_M(\sigma)(v) \\
\iff &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \land m \notin \{m'' \in From\_Sender(v,\sigma) : Later\_From(m'',v,\sigma) = \emptyset\} \\
\iff &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \land Later\_From(m,v,\sigma) \neq \emptyset \\
\iff &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \land \exists m^* \in Later\_From(m,v,\sigma) \\
\iff &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \land \exists m^* \in Later\_From(m,v,\sigma), m^* \in Later(m,\sigma) \cap From\_Sender(v,\sigma) \\
\iff &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \land \exists m^* \in Later\_From(m,v,\sigma), m^* \in Later(m,\sigma) \\
\iff &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \\
&\quad \land \exists m^* \in Later\_From(m,v,\sigma), m^* \in Later(m,\sigma) \land m \in Justification(m^*) \\
\implies &\forall m' \in From\_Sender(v,\sigma), m \preceq m' \\
&\quad \land \exists m^* \in Later\_From(m,v,\sigma), Justification(m) \subseteq Justification(m^*) \land m \in Justification(m^*) \\
\implies &\forall m' \in From\_Sender(v,\sigma), |Justification(m)| \geq |Justification(m'))| \\
&\quad \land \exists m^* \in Later\_From(m,v,\sigma), Justification(m) \subseteq Justification(m^*) \land m \in Justification(m^*) \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), Justification(m) \subseteq Justification(m^*) \land m \in Justification(m^*) \\
&\quad \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), Justification(m) \subseteq Justification(m^*) \land \{m\} \subseteq Justification(m^*) \\
&\quad \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), Justification(m) \cup \{m\} \subseteq Justification(m^*) \\
&\quad \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), |Justification(m)| + |\{m\}| - |Justification(m) \cap \{m\}| \leq Justification(m^*) \\
&\quad \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), |Justification(m)| + |\{m\}| - |Justification(m) \cap \{m\}| \leq Justification(m^*) \\
&\quad \land Justification(m) \cap \{m\} = \emptyset \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), |Justification(m)| + |\{m\}| - |Justification(m) \cap \{m\}| \leq Justification(m^*) \\
&\quad \land |Justification(m) \cap \{m\}| = 0 \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), |Justification(m)| + |\{m\}| \leq Justification(m^*) \\
&\quad \land |Justification(m)| \geq |Justification(m^*)| \\
\implies &\exists m^* \in Later\_From(m,v,\sigma), |Justification(m)| + |\{m\}| \leq Justification(m^*) \land |\{m\}| \geq 1 \\
&\quad \land |Justification(m)| \geq |Justification(m^*)|
\end{aligned}
$$

This leads to a contradiction in these three inequalities. Therefore, a minimal message in $From\_Sender(v,\sigma)$ is a latest message from validator v. ■

Lemma 9. There is at least one minimal element in $(From\_Sender(v,\sigma), \preceq)$ for a $v \in Observed(\sigma)$.

Proof.

$$
\begin{aligned}
&v \in Observed(\sigma) \\
\implies &v \in \{Sender(m) : m \in \sigma\} \\
\implies &\exists m \in \sigma, Sender(m) = v \\
\implies &\exists m \in \{m \in \sigma : Sender(m) = v\} \\
\implies &\exists m \in From\_Sender(v,\sigma) \\
\implies &From\_Sender(v,\sigma) \neq \emptyset
\end{aligned}
$$

By Well-Ordering Principle, a non-empty countable total order always has a minimal element. ■

Hence, observed validators have latest messages, i.e., $\forall \sigma \in \Sigma, \forall v \in \mathcal{V}$,
$v \in Observed(\sigma) \implies |L_M(\sigma)(v)| \geq 1$

Lemma 10 (Observed non-equivocating validators have one latest messages)**.** $\forall \sigma \in \Sigma, \forall v \in \mathcal{V}$

$$v \in Observed(\sigma) \land v \notin E(\sigma) \implies |L_M(\sigma)(v)| = 1$$

Proof.

$$\begin{aligned} & v \in Observed(\sigma) \wedge v \notin E(\sigma) \\ \implies & |L_M(\sigma)(v)| \geq 1 \wedge |L_M(\sigma)(v)| \leq 1 \\ \implies & |L_M(\sigma)(v)| = 1 \end{aligned}$$

■

Definition 4.11 (Latest messages from non-Equivocating validators)**.**

$$L_M^H : \Sigma \to (\mathcal{V} \to \mathcal{P}(M))$$

$$L_M^H(\sigma)(v) = \begin{cases} \emptyset & \text{for } v \in E(\sigma) \\ L_M(\sigma)(v) & \text{otherwise} \end{cases}$$

Note that the map returned by this function has values $L_M^H(\sigma)(v) = \emptyset$ for any validators who are equivocating in σ or who don't have any messages in σ.

Definition 4.12 (Latest honest message driven estimator)**.**

$$Latest\_Honest\_Message\_Driven : \mathcal{P}(C)^{\Sigma} \to \{True, False\}$$

$$Latest\_Honest\_Message\_Driven(\mathcal{E}) :\Leftrightarrow \exists \hat{\mathcal{E}} \in \mathcal{P}(C)^{\mathcal{P}(M)^{\mathcal{V}}}, \quad \mathcal{E} = \hat{\mathcal{E}} \circ L_M^H$$

Definition 4.13 (Latest Estimates from non-Equivocating validators)**.**

$$L_E^H : \Sigma \to (\mathcal{V} \to \mathcal{P}(\mathcal{C}))$$

$$L_E^H(\sigma)(v) = \{Estimate(m) : m \in L_M^H(\sigma)(v)\}$$

As above, $L_E^H(\sigma)(v) = \emptyset$ for validators v who are equivocating or missing in σ.

Definition 4.14 (Latest honest estimate driven estimator)**.**

$$Latest\_Honest\_Message\_Driven : \mathcal{P}(C)^{\Sigma} \to \{True, False\}$$

$$Latest\_Honest\_Message\_Driven(\mathcal{E}) :\Leftrightarrow \exists \hat{\mathcal{E}} \in \mathcal{P}(C)^{\mathcal{P}(M)^{\mathcal{V}}}, \quad \mathcal{E} = \hat{\mathcal{E}} \circ L_E^H$$

All the example protocols we will give will have latest honest estimate driven estimators.

4.2 Casper the Friendly Binary Consensus

Definition 4.15 (Argmax)**.**

$$Argmax : \mathcal{P}(a) \times (a \to \mathbb{R}) \to \mathcal{P}(a)$$

$$Argmax(X, f) = \{x \in X : \nexists x' \in X, f(x') > f(x)\}$$

where a is a type variable.

For notational convenience, we may write $Argmax(X, f)$ as $\arg\max_{x \in X} f(x)$.

Definition 4.16 (Score)**.**

$$Score : \{0, 1\} \times \Sigma \to \mathbb{R}$$

$$Score(x, \sigma) = \sum_{v \in V : x \in L_E^H(\sigma)(v)} \mathcal{W}(v)$$

Definition 4.17 (Casper the Friendly Binary Consensus)**.**

$$\mathcal{C} = \{0, 1\}$$

$$\mathcal{E}(\sigma) = \arg\max_{c \in C} \ Score(c, \sigma)$$

Definition 4.18 (Example non-trivial properties of this binary consensus protocol)**.**

$$P = \{p \in P_{\mathcal{C}} : \exists! c \in C, p(c) = True\}$$

4.3 Casper the Friendly Integer Consensus

Definition 4.19 (Weighted Median)**.**

$$Median : \mathcal{P}(\mathbb{Z}) \times (\mathbb{Z} \to \mathbb{R}) \to \mathcal{P}(\mathbb{Z})$$
$$Median(X, W) = \{x \in X : \sum_{x' \in X : x' < x} W(x') \leq \sum_{x' \in X} W(x')/2 \\ \wedge \sum_{x' \in X : x' > x} W(x') \leq \sum_{x' \in X} W(x')/2\}$$

Definition 4.20 (Score)**.**

$$Score : \mathbb{Z} \times \Sigma \to \mathbb{R}$$
$$Score(x, \sigma) = \sum_{v \in V : x \in L_E^H(\sigma)(v)} \mathcal{W}(v)$$

Definition 4.21 (Casper the Friendly Integer Consensus)**.**

$$\mathcal{C} = \mathbb{Z}$$
$$\mathcal{E}(\sigma) = Median(\cup_{v \in \mathcal{V}} L_E^H(\sigma)(v), \lambda x.Score(x, \sigma))$$

Definition 4.22 (Example non-trivial properties of the integer consensus protocol)**.**

$$P = \{p \in P_{\mathcal{C}} : \exists! z \in \mathcal{C}, p(z) = True\}$$

4.4 Casper the Friendly GHOST

Greedy Heaviest Observed Sub-Tree[3] is a contruction proposed to tackle reduced security issues in blockchains with fast conformation times. Here, we present a CBC specification for a GHOST-based blockchain fork choice rule.

Starting from a genesis block g, we can define all blocks in a blockchain to have a previous block and some block data D.

Definition 4.23 (Blocks)**.**

$$B_0 = \{g\}$$
$$B_n = B_{n-1} \times D$$
$$B = \bigcup_{i=0}^{\infty} B_i$$

Blocks/blockchains will be the consensus value for blockchain consensus.

Every block in a blockchain has a single previous block.

Definition 4.24 (Previous block resolver)**.**

$$Prev : B \to B$$
$$Prev(b) = \begin{cases} g & \text{if } b = g \\ b' & \text{otherwise, if } b = (b', d) \end{cases}$$

$$Prev(b) = \begin{cases} g & \text{if } b = g \\ Proj_1(b) & \text{otherwise} \end{cases}$$

Definition 4.25 (n-cestor: n'th generation ancestor block)**.**

$$n\text{-}cestor : B \times \mathbb{N} \to B$$
$$n\text{-}cestor(b, n) = \begin{cases} b & \text{if } n = 0 \\ n\text{-}cestor(Prev(b), n-1) & \text{otherwise} \end{cases}$$

A block is "in the blockchain" of another block if it one of its ancestors.

Definition 4.26 (Blockchain membership, $m_1 \lfloor m_2$)**.**

$$\cdot \lfloor \cdot : B \times B \to \{True, False\}$$
$$b_1 \lfloor b_2 :\Leftrightarrow \exists n \in \mathbb{N}, b_1 = n\text{-}cestor(b_2, n)$$

We define the "score" of a block b in state σ as the total weight of validators with latest blocks b' such that $b \lfloor b'$.

Definition 4.27 (Score of a block)**.**

$$Score : M \times \Sigma \to \mathbb{R}$$
$$Score(b, \sigma) = \sum_{v \in \mathcal{V} : \exists b' \in L_E^H(\sigma)(v), b \lfloor b'} \mathcal{W}(v)$$

The "children" of a block b in a protocol state σ are the blocks with b as their prevblock.

Definition 4.28.

$$Children : M \times \Sigma \to \mathcal{P}(M)$$
$$Children(b, \sigma) = \{b' \in \bigcup_{m \in \sigma} \{Estimate(m)\} : Prev(b') = b\}$$

We now have the language required to define the estimator for the blockchain consensus, the Greedy Heaviest-Observed Sub-Tree fork choice rule, or GHOST!

Definition 4.29.

$$Best\_Children : B \times \Sigma \to \mathcal{P}(B)$$
$$Best\_Children(b, \sigma) = \operatorname*{arg\,max}_{b' \in Children(b,\sigma)} Score(b', \sigma)$$

Definition 4.30.

$$GHOST : \mathcal{P}(B) \times \Sigma \to \mathcal{P}(B)$$
$$GHOST(\underline{b}, \sigma) = \bigcup_{\substack{b \in \underline{b} : \\ Children(b,\sigma) \neq \emptyset}} GHOST(Best\_Children(b, \sigma), \sigma) \cup \bigcup_{\substack{b \in \underline{b} : \\ Children(b,\sigma) = \emptyset}} \{b\}$$

Definition 4.31 (Casper the Friendly Ghost)**.**

$$\mathcal{C} = B$$
$$\mathcal{E}(\sigma) = GHOST(\{g\}, \sigma)$$

Definition 4.32 (Example non-trivial properties of consensus values)**.**

$$P = \{p \in P_{\mathcal{C}} : \exists! b \in \mathcal{C}, \forall b' \in \mathcal{C}, (b \lfloor b' \implies p(b') = True) \wedge (\neg(b \lfloor b') \implies p(b') = False)\}$$

4.5 Casper the Friendly CBC Finality Gadget

Finality gadgets are consensus protocols on the blocks of an underlying blockchain, which has its own block structure and fork choice rule. Casper the Friendly Finality Gadget[4] describes one such construction of a finality gadget.

Here, we will describe the underlying blockchain and the conditions it must satisfy, before showing how to layer a CBC Casper finality gadget on top, namely in the form of a change to the blockchains forkchoice.

Definition 4.33 (The underlying blockchain)**.** We assume the blockchain has blocks:

Starting from a genesis block g, we can define all blocks in a blockchain to have a previous block and some block data D.

Definition 4.34 (Blocks)**.**

$$\begin{aligned} B_0 &= \{g\} \\ B_n &= B_{n-1} \times D \\ B &= \bigcup_{i=0}^{\infty} B_i \end{aligned}$$

Blocks/blockchains will be the consensus value for blockchain consensus.

Every block in a blockchain has a single previous block.

Definition 4.35 (Previous block resolver)**.**

$$Prev : B \to B$$

$$Prev(b) = \begin{cases} g & \text{if } b = g \\ b' & \text{otherwise, if } b = (b', d) \end{cases}$$

$$Prev(b) = \begin{cases} g & \text{if } b = g \\ Proj_1(b) & \text{otherwise} \end{cases}$$

Definition 4.36 (n-cestor: n'th generation ancestor block)**.**

$$n\text{-}cestor : B \times \mathbb{N} \to B$$

$$n\text{-}cestor(b, n) = \begin{cases} b & \text{if } n = 0 \\ n\text{-}cestor(Prev(b), n-1) & \text{otherwise} \end{cases}$$

A block is "in the blockchain" of another block if it one of its ancestors.

Definition 4.37 (Blockchain membership, $m_1 \lfloor m_2$)**.**

$$\begin{aligned} \cdot \lfloor \cdot &: B \times B \to \{True, False\} \\ b_1 \lfloor b_2 &:\Leftrightarrow \exists n \in \mathbb{N}, b_1 = n\text{-}cestor(b_2, n) \end{aligned}$$

We note that $\forall b \in B, g \lfloor b$.

The blockchain is also equipped with a height map that satisfies the following conditions:

$$Height : B \to \mathbb{N}$$

$$Height(b) = \begin{cases} 0 & \text{if } b = g \\ 1 + Height(Prev(b)) & \text{otherwise} \end{cases}$$

Finally, the underlying blockchain has a forkchoice rule, which is parametric in a starting block (where the forkchoice "begins"):

$$\mathcal{F} : B \times \mathcal{P}(B) \to B$$

The forkchoice returns a block "on top of" the starting block:

$$\forall b \in B, \forall \underline{B} \in \mathcal{P}(B), b \lfloor \mathcal{F}(b, \underline{B})$$

We can now construct the consensus values for the finality gadget.

Definition 4.38 (Consensus values in the CBC Finality Gadget)**.** Finality gadgets specify 'epoch lengths,' which is essentially how frequently they operate:

$$Epoch\_Length \in \mathbb{N}_+$$

The consensus values are blocks on the epoch boundries:

$$\mathcal{C} = \{b \in B : Height(b) \equiv 0 \pmod{Epoch\_Length}\}$$

We now construct the finality gadgets new forkchoice, which can be understood as GHOST on the epochs, followed by the underlying blockchain's forkchoice from the tip epoch.

As $\mathcal{C}$ is a subset of B, we can inherent the blockchain membership relation $\lfloor$ from B.

This allows us to define the score of an epoch.

Definition 4.39 (Epoch Score)**.**

$$Epoch\_Score : \mathcal{C} \times \Sigma \to \mathbb{R}$$
$$Epoch\_Score(e, \sigma) = \sum_{v \in \mathcal{V} : \exists b' \in L_E^H(\sigma)(v), b \lfloor b'} \mathcal{W}(v)$$

Definition 4.40 (Children Epochs)**.**

$$Children\_Epochs : \mathcal{C} \times \Sigma \to \mathcal{C}$$
$$Children\_Epochs(e, \sigma) = \{b' \in Blocks\_In(\sigma) : e \lfloor b' \wedge Height(b') = Height(e) + Epoch\_Length\}$$

Definition 4.41.

$$Best\_Children\_Epochs : C \times \Sigma \to \mathcal{P}(C)$$
$$Best\_Children\_Epochs(e, \sigma) = \underset{e' \in Children\_Epochs(e,\sigma)}{\arg\max} Epoch\_Score(e', \sigma)$$

We can now define GHOST on the epochs:

Definition 4.42.

$$GHOST : \mathcal{P}(C) \times \Sigma \to \mathcal{P}(B)$$
$$GHOST(\underline{b}, \sigma) = [\bigcup_{\substack{\underline{b} \in \underline{b} \\ Children\_Epochs(b,\sigma) \neq \emptyset}} GHOST(Best\_Children\_Epochs(b, \sigma), \sigma)]$$
$$\cup [\bigcup_{\substack{b \in \underline{b} \\ Children\_Epochs(b,\sigma) = \emptyset}} \{b\}]$$

Definition 4.43 (Estimator for the CBC Finality gadget)**.**

$$\mathcal{E}(\sigma) = \{b \in \mathcal{C} : \exists b' \in GHOST(\{g\}, \sigma), b' \lfloor b\}$$

Definition 4.44 (Fork Choice Rule for the CBC Finality gadget)**.** Original fork choice rule:

$$\mathcal{F}(\{g\}, \underline{b})$$

New fork choice rule:

$$\mathcal{F}(\mathcal{E}(\sigma), \underline{b}))$$

Finally, we now insist that the forkchoice rule of the underlying chain starts at the estimator of the finality gadget, completing our definition of the underlying blockchain protocol.

4.6 Casper the Friendly CBC Sharded Blockchain

Definition 4.45 (Shard IDs S)**.**

$$S$$

Definition 4.46 (Message Payloads P)**.**

$$P$$

Definition 4.47 (Block Data D)**.**

$$D$$

Definition 4.48 (Lists of Things)**.** For a set X, let X^* denote all the finite length lists of elements in set X.

Definition 4.49 (List Prefix $\preceq$)**.** $A \preceq B$ iff list A is a prefix of list B

We note that the blocks we will construct for the sharded blockchain satisfy this equation:

$$B \subseteq S \times B \times (S \to (B \times \mathbb{N} \times P)^*) \times (S \to (B \times \mathbb{N} \times P)^*) \times (S \to B \cup \{\emptyset\}) \times D$$

which allows us to define the following convenience functions:

Definition 4.50 (Blocks B)**.**

$$\begin{aligned} \text{If } b = (\mathit{shard\_id},\ \mathit{prev\_blk},\ \mathit{sent\_log},\ \mathit{recv\_log},\ \mathit{src},\ \mathit{blk\_data}) \in B \text{ , then} \\ \mathit{Shard}(b) = \mathit{shard\_id} \\ \mathit{Prev}(b) = \mathit{prev\_blk} \\ \mathit{Sent\_Log}(b) = \mathit{sent\_log} \\ \mathit{Received\_Log}(b) = \mathit{recv\_log} \\ \mathit{Source}(b) = \mathit{src} \end{aligned}$$

Definition 4.51 (Cross-shard Messages Q)**.**

$$\begin{aligned} Q \subseteq B \times \mathbb{N} \times P \\ \text{If } (b, n, d) \in Q \text{ , then:} \\ \mathit{Base}((b, n, d)) = b \\ \mathit{TTL}((b, n, d)) = n \end{aligned}$$

Definition 4.52 (Genesis Logs)**.**

$$\begin{aligned} \mathit{Log}_g : S \to (B \times \mathbb{N} \times P)^* \\ \forall s \in S, \mathit{Log}_g(s) = [] \end{aligned}$$

Definition 4.53 (Genesis Sources)**.**

$$\begin{aligned} \mathit{Sources}_g : (S \to (B \times \mathbb{N} \times P)^*) \times (S \to B \cup \{\emptyset\}) \\ \forall s \in S, \mathit{Sources}_g(s) = \emptyset \end{aligned}$$

4.6.1 Block Validity Conditions

We now introduce further restrictions on blocks, in the form of validity conditions. As these are functions on single blocks, we note that the type signature of these conditions is $B \to \{True, False\}$.

Definition 4.54 (Shard ID Consistency)**.**

$$\begin{aligned} \mathit{Shard\_ID\_Consistency}(b) :\Leftrightarrow & \mathit{Shard}(\mathit{Prev}(b)) = \mathit{Shard}(b) \wedge \\ & \forall s \in S, \big[\mathit{Shard}(\mathit{Sources}(b)(s)) = s \\ & \quad \wedge \forall q \in \mathit{Sent\_Log}(b)(s), \mathit{Shard}(\mathit{Base}(q)) = s \\ & \quad \wedge \forall q \in \mathit{Received\_Log}(b)(s), \mathit{Shard}(\mathit{Base}(q)) = \mathit{Shard}(b)\big] \end{aligned}$$

Definition 4.55 (Monotonicity Conditions)**.**
Monotonic Sources

$$Source\_Monotonicity(b) :\Leftrightarrow \forall s \in S, Source(b)(s) \upharpoonright Source(Prev(b))(s)$$

Sent Log Monotonicity

$$Sent\_Log\_Monotonicity(b) :\Leftrightarrow \forall s \in S, Sent\_Log(Prev(b))(s) \preceq Sent\_Log(b)(s)$$

Receive Log Monotonicity

$$Received\_Log\_Monotonicity(b) :\Leftrightarrow \forall s \in S, Received\_Log(Prev(b))(s) \preceq Received\_Log(b)(s)$$

Monotonic Bases

$$\begin{aligned} Monotonic\_Sent\_Bases(b) :\Leftrightarrow \forall s \in S, \forall i \in [2, Length(Sent\_Log(b)(s))], \\ Base(Sent\_Log(b)(s)[i-1]) \upharpoonright Base(Sent\_Log(b)(s)[i]) \\ Monotonic\_Received\_Bases(b) :\Leftrightarrow \forall s \in S, \forall i \in [2, Length(Received\_Log(b)(s))], \\ Base(Received\_Log(b)(s)[i-1]) \upharpoonright Base(Received\_Log(b)(s)[i]) \end{aligned}$$

Base-Source Monotonicity

$$Base\_Source\_Monotonicity(b) :\Leftrightarrow \forall s \in S, \forall q \in Sent\_Log(b)(s), Source(b)(s) \upharpoonright Base(q)$$

Definition 4.56 (Messsage Arrival Conditions)**.**
Messages Sent by Source are Received Once and In-Order

$$Receive\_Once\_In\_Order(b) :\Leftrightarrow \forall s \in S, Received\_Log(b)(s) \preceq Sent\_Log(Source(b)(s))(Shard(b))$$

Receive Messages by Expiry

$$\begin{aligned} Receive\_By\_Expiry(b) :\Leftrightarrow \forall s \in S, \forall q \in Received\_Log(b)(s), \\ Base(q) \upharpoonright b \wedge Height(b) - Height(Base(q)) \leq TTL(q) \end{aligned}$$

Messages Sent but Not Received are Not Expired

$$\begin{aligned} Unreceived\_Messages\_Unexpired(b) :\Leftrightarrow \forall s \in S, \forall q \in Sent\_Log(b)(s) - Received\_Log(Source(b)(s))(Shard(b)), \\ Height(Source(b)(s)) - Height(Base(q)) < TTL(q) \end{aligned}$$

We now have enouch structure to define the blocks in a sharded blockchain.

4.6.2 Blocks

Definition 4.57 (Set of Blocks B)**.**

$$
\begin{aligned}
B^0 &= \bigcup_{i \in S} (i \times \{\emptyset\} \times Log_g \times (Log_g \times Received\_Sources_g) \times Genesis\_Data) \\
B^n &= \Big\{ b \in S \times B^{n-1} \times (S \to (B^{n-1} \times \mathbb{N} \times P)^*) \times [(S \to (B^{n-1} \times \mathbb{N} \times P)^*) \times (S \to B^{n-1} \cup \{\emptyset\})] \times D : \\
&\qquad Shard\_ID\_Consistency(b) \wedge \\
&\qquad \big[Source\_Monotonicity(b) \\
&\qquad \wedge Sent\_Log\_Monotonicity(b) \\
&\qquad \wedge Received\_Log\_Monotonicity(b) \\
&\qquad \wedge Monotonic\_Sent\_Bases(b) \\
&\qquad \wedge Monotonic\_Received\_Bases(b) \\
&\qquad \wedge Base\_Source\_Monotonicity(b)\big] \wedge \\
&\qquad \big[Receive\_Once\_In\_Order(b) \\
&\qquad \wedge Receive\_By\_Expiry(b) \\
&\qquad \wedge Unreceived\_Messages\_Unexpired(b)\big] \\
&\Big\} \\
B &= \bigcup_{n=0}^{\infty} B^n
\end{aligned}
$$

So we have the following consensus values

Definition 4.58 (Consensus values)**.**

$$\mathcal{C} = B$$

4.6.3 Estimator

For this presentation we will assume that we have two shards $S = \{P, C\}$, where P is the "parent" (or root) shard and C is the "child" shard. The "child"'s estimates will "follow" the parents. The parent shard's fork choice rule is independent of the child's, and is (as an example for the sake of this presentation) is simply GHOST:

Definition 4.59 (Fork Choice $\mathcal{E}_P$ (for root shard))**.**

$$
\begin{aligned}
&\mathcal{E}_P : \Sigma \to B \\
&\mathcal{E}_P(\sigma) = GHOST(\{g_P\}, \sigma)
\end{aligned}
$$

The forkchoice of the parent "restricts" the fork choices that are allowed for the child shard. Through this restriction, fork choices on the parent and child shard will be "synchronized", in that they won't have sources that disagree with blocks chosen on other shards, and there won't be any expired unreceived messages from the point of view of the blocks returned by the fork choice.

To maintain these conditions, we introduce 4 "filter conditions", which is a condition that will be used to prevent blocks that don't synchronize with the parent fork choice to be chosen in the child's fork choice. The condition is defined to be parametric in a block from the parent shard b_P.

Definition 4.60 (Filter Conditions)**.**
Parent to Child Source Block Consistency

$$Filter_1(b_C; b_P) :\Leftrightarrow Source(b_P)(C) \not| b_C$$

Child to Parent Source Block Consistency

$$Filter_2(b_C; b_P) :\Leftrightarrow Source(b_C)(P) \not\downarrow b_P$$

We note that the relation $\downarrow$ holds for two block b, b' if $b \downarrow b' \vee b' \downarrow b$

Parent to Child Sent Message Expiry Condition

$$Filter_3(b_C; b_P) :\Leftrightarrow \exists q \in Sent\_Log(b_P)(C), q \notin Received\_Log(b_C)(P) \wedge Height(Base(q)) + TTL(q) \leq Height(b_C)$$

Child to Parent Sent Message Expiry Condition

$$Filter_4(b_C; b_P) :\Leftrightarrow \exists q \in Sent\_Log(b_C)(P), q \notin Received\_Log(b_P)(C) \wedge Height(Base(q)) + TTL(q) \leq Height(b_P)$$

Total Filter

$$Filter\_Block(b_C; b_P) :\Leftrightarrow Filter_1(b_C; b_P) \vee Filter_2(b_C; b_P C) \vee Filter_3(b_C; b_P) \vee Filter_4(b_C; b_P)$$

Definition 4.61 (Filtered Children)**.**

$$Filtered\_Children : B \times \mathcal{P}(B) \times (B \to \{True, False\}) \to \mathcal{P}(B)$$
$$Filtered\_Children(b, blocks, filter\_block) = \{b' \in blocks : Prev(b') = b \wedge \neg Filter\_Block(b'; filter\_block)\}$$

Definition 4.62 (Best Child)**.**

$$Best\_Children : B \times \mathcal{P}(B) \times (B \to \mathbb{R}) \times B \to \mathcal{P}(B)$$
$$Best\_Children(b, blocks, W, filter\_block) = \underset{b' \in Filtered\_Children(b, blocks, filter\_block)}{\arg\max} W(b')$$

Definition 4.63 (Block score)**.**

$$Score : \Sigma \to (B \to \mathbb{R}^+)$$
$$Score(\sigma)(b) = \sum_{v \in \mathcal{V} : \exists b' \in L_E^H(\sigma)(v), b \downarrow b'} \mathcal{W}(v)$$

We note that this score function works properly when each validator is assigned to a specific shard. In the future, this score function will count the score with respect to the latest honest messages on a specific shard.

Definition 4.64.

$$Filtered\_GHOST : \mathcal{P}(B) \times \mathcal{P}(B) \times (B \to \mathbb{R}) \times B \to \mathcal{P}(B)$$
$$Filtered\_GHOST(\underline{b}, blocks, W, filter\_block) =$$
$$\bigcup_{\substack{b \in \underline{b} : \\ Best\_Children(b, blocks, W, filter\_block) \neq \emptyset}} Filtered\_GHOST(Best\_Children(b, blocks, W, filter\_block), blocks, W, filter\_block) \cup$$
$$\bigcup_{\substack{b \in \underline{b} : \\ Best\_Children(b, blocks, W, filter\_block) = \emptyset}} \{b\}$$

Now, we can finially construct the forkchoice of the child, which can be understood as GHOST run from the sourc block on the child, where any blocks that satisfy the filter conditions are not chosen by GHOST.

Definition 4.65 (Fork Choice $\mathcal{E}_C$ (for child shard C))**.**

$$\mathcal{E}_C : \Sigma \to \mathcal{P}(B)$$
$$\mathcal{E}_C(\sigma) = Filtered\_GHOST(\{Source(E_P(\sigma))(C)\}, Blocks(\sigma), Score(\sigma), E_P(\sigma))$$

The estimator for the CBC sharded blockchain consensus protocol gives fork choices for both the parent and the child:

Definition 4.66 (Estimator $\mathcal{E}$)**.**

$$\mathcal{E}_C : \Sigma \to \mathcal{P}(B)$$
$$\mathcal{E}(\sigma) = E_P(\sigma) \cup E_C(\sigma)$$

5 Discussion and Conclusion

In summary, we've partially specified a family of consensus protocols and shown that every member of this family has asynchronous, Byzantine fault tolerant consensus safety. Furthermore, we've given examples of members of the family of increasing complexity to demonstrate the power and flexibility of our approach to correct-by-construction consensus protocol design.

Although the protocols given here are not fully enough defined to serve as production systems, we contend that the specifications given here make it easy to define many practical protocols. For example, we have been able to show that it's possible to do consensus with asynchronous Byzantine fault tolerant safety with the same network overhead as Nakamoto consensus. We strongly suspect that CBC Casper protocols are qualitatively different from traditional consensus protocols like Paxos[5] or PBFT[6].

Notable related work that isn't discussed here includes work on fault attribution and classification, the detection of decisions in CBC Casper protocols, light client CBC Casper protocols, running CBC Casper with with extra-protocol fault tolerance thresholds, doing validator set changes, guaranteeing the validity and availability of messages, and work on ensuring the existence of incentives in proof-of-stake.

Work on liveness, validator strategies for making consensus messages, and for preventing denial of service attacks is ongoing. Furthermore, work of parametrizing penalties and rewards for security-deposit based proof-of-stake is in relatively early stages.

Exploration of the CBC Casper family of protocols (and their implementations) is in early stages, and we have relatively very little undertanding of what would make an "ideal" estimator, or of what kind of estimators lead to the lowest latency or lowest overhead decisions. For example, Vitalik's discussion of latest message based estimators vs "immediate" message driven fork choice rules[7].

The work shared here is supposed to be the simplest formalization that will give the reader and understanding of the logic behind the proof of asynchronous Byzantine fault tolerant safety for the broadest family of CBC Casper consensus protocols. It isn't meant to be a complete specification of a consensus protocol, nor does the specification represent the best way to implement even this partial specification. We therefore primarily hope that readers find this work to be educational, and secondarily that it puts them in a position to contribute to "CBC Casper research".

References

[1] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the Asscktion for Computing Machinery*, vol. 32, no. 2, pp. 374–382, April 1985. [Online]. Available: https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash systems," 11 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," 12 2013. [Online]. Available: https://eprint.iacr.org/2013/881.pdf

[4] Vitalik Buterin and Virgil Griffith, "Casper the Friendly Finality Gadget," 2017. [Online]. Available: https://arxiv.org/pdf/1710.09437.pdf

[5] R. V. RENESSE and D. ALTINBUKEN, "Paxos made moderately complex," *ACM Comput. Surv.*, vol. 47, no. 3, April 2015, article no. 42. [Online]. Available: http://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf

[6] Miguel Castro and Barbara Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999. [Online]. Available: http://pmg.lcs.mit.edu/papers/osdi99.pdf

[7] Vitalik Buterin. [Online]. Available: https://twitter.com/vitalikbuterin/status/1029900695925706753?lang=en

6 Appendix

Here we give two concise equivalent definitions of the Minimal CBC Casper protocol states definition. They are less readable than the presentation given in the paper, but might be of interests to some readers.

Both of the definitions given here use the same "parameters" as presented in this draft.

$$\begin{aligned}
&\text{Consensus Values :} && \mathcal{C} \in SET \ \wedge \ \mathcal{C} \neq \emptyset && (101)\\
&\text{Validators :} && \mathcal{V} \in SET \ \wedge \ \mathcal{V} \neq \emptyset && (102)\\
&\text{Estimator :} && \mathcal{E} : SET \to \mathcal{P}(\mathcal{C}) && (103)\\
&\text{Validator Weights :} && \mathcal{W} : \mathcal{V} \to \mathbb{R}_+ && (104)\\
&\text{Equivocation Fault Tolerance Threshold :} && t \in \mathbb{R} \wedge 0 \leq t < \sum_{v \in \mathcal{V}} \mathcal{W}(v) && (105)
\end{aligned}$$

The first definition is almost exactly the definition given in the draft, but it in-lines definitions and uses pattern matching. Instead of defining intermediate states Σ and then actual protocol states Σ_t, we directly define $\Gamma = \Sigma_t$:

$$\begin{aligned}
\Gamma_0 &= \{\emptyset\} && (106)\\
M_n &= \{(c, v, \gamma) \in \mathcal{C} \times \mathcal{V} \times \Gamma_n : c \in \mathcal{E}(\gamma)\} && (107)\\
\Gamma_n &= \{\gamma \in \mathcal{P}_{fin}(M_{n-1}) : (c, v, \gamma') \in \gamma \implies \gamma' \subseteq \gamma && (108)\\
&\qquad \wedge \sum_{\substack{v \in \mathcal{V}: \\ \exists (c_1,v,\gamma_1),(c_2,v,\gamma_2) \in \gamma^2, \\ c_1 \neq c_2 \vee \gamma_1 \neq \gamma_2 \\ (c_2,v,\gamma_2) \notin \gamma_1 \ \wedge \ (c_1,v,\gamma_1) \notin \gamma_2}} \mathcal{W}(v) \leq t\} \qquad \text{for } n > 0 && (109)\\
M &= \bigcup_{n=0}^{\infty} M_n && (110)\\
\Gamma &= \bigcup_{n=0}^{\infty} \Gamma_n && (111)\\
& && (112)
\end{aligned}$$

The second definition is as a fixed point of a function/action that adds protocol messages to protocol states (but only adds them if the fault weight is not pushed beyond the equivoation fault tolerance threshold t), namely:

$$sf(X) = X \cup \bigcup_{\substack{(\gamma,\gamma') \in X^2: \\ \gamma \subseteq \gamma' \\ c \in \mathcal{E}(\gamma) \\ v \in \mathcal{V}}} \begin{cases} \{\gamma' \cup \{(c, v, \gamma)\}\} & \text{, if } \sum_{\substack{v' \in \mathcal{V}: \\ \exists (c_1,v',\gamma_1),(c_2,v',\gamma_2) \in (\gamma' \cup \{(c,v,\gamma)\})^2 \\ c_1 \neq c_2 \ \vee \ \gamma_1 \neq \gamma_2 \\ (c_2,v',\gamma_2) \notin \gamma_1 \wedge (c_1,v',\gamma_1) \notin \gamma_2}} \mathcal{W}(v') \leq t \\ \{\gamma'\} & \text{, otherwise} \end{cases} \tag{113}$$

The protocol states can be defined as the following fixed point, which the result of applying our function f to "initial" protocol state $\emptyset$ a countable infinite number of times.

$$\Gamma = \ \text{fix } f \ \{\emptyset\} = f(f(f(f(f(\dots f(f(\{\emptyset\})) \dots)))))$$

And the messages are the elements of the protocol states:

$$M = \bigcup_{\substack{\gamma \in \Gamma \\ m \in \gamma}} \{m\}$$