Casper the Friendly Finality Gadget: Basic Structure

Vitalik Buterin
Ethereum Foundation

August 15, 2017

Abstract

We give an introduction to the non-economic details of Casper: the Friendly Finality Gadget, Phase 1.

1 Introduction, Protocol I

In the Casper protocol, there is a set of validators, and in each epoch validators have the ability to send two kinds of messages:

$$[PREPARE, epoch, hash, epoch_{source}, hash_{source}]$$

and

$$[COMMIT, epoch, hash]$$

An *epoch* is a period of 100 epochs; epoch n begins at block $n * 100$ and ends at block $n*100+99$. A *checkpoint for epoch* n is a block with number $n*100-1$; in a smoothly running blockchain there will usually be one checkpoint per epoch, but due to network latency or deliberate attacks there may be multiple competing checkpoints. The *parent checkpoint* of a checkpoint is the 100th ancestor of the checkpoint block, and an *ancestor checkpoint* of a checkpoint is either the parent checkpoint, or an ancestor checkpoint of the parent checkpoint. We define the *ancestry hash* of a checkpoint as follows:

- The ancestry hash of the implied "genesis checkpoint" before epoch 0 is zero.

- The ancestry hash of any other checkpoint is the keccsk256 hash of the ancestry hash of its parent concatenated with the hash of the checkpoint.

Ancestry hashes thus form a direct hash chain, and otherwise have a one-to-one correspondence with checkpoint hashes.

During epoch n, validators are expected to send prepare and commit messages specifying epoch n, and the ancestry hash of a checkpoint for epoch n (i.e. with block number $n * 100 - 1$). Prepare messages are expected to specify as $hash_{source}$ a checkpoint for any previous epoch which is *justified* (see below), and the $epoch_{source}$ is expected to be the epoch of that checkpoint.

Each validator has a *deposit size*; when a validator joins their deposit size is equal to the number of coins that they deposited, and from there on each validator's deposit size rises and falls as the validator receives rewards and penalties. For the rest of this paper, when we say "$\frac{2}{3}$ of validators", we are referring to a *deposit-weighted* fraction; that is, a set of validators whose combined deposit size equals to at least $\frac{2}{3}$ of the total deposit size of the entire set of validators. We also use "$\frac{2}{3}$ commits" as shorthand for "commits from $\frac{2}{3}$ of validators".

If, during an epoch e, for some specific ancestry hash h, for any specific ($epoch_{source}, hash_{source}$ pair), there exist $\frac{2}{3}$ prepares of the form

$$[PREPARE, e, h, epoch_{source}, hash_{source}]$$

, then h is considered *justified.* If $\frac{2}{3}$ commits are sent of the form

$$[COMMIT, e, h]$$

then h is considered *finalized.*

We add the following modifications:

- For a checkpoint to be finalized, it must be justified.
- For a checkpoint to be justified, the $hash_{source}$ used to justify it must itself be justified.
- Prepare and commit messages are only accepted as part of blocks; that is, for a client to see $\frac{2}{3}$ commits of some hash, they must receive a block such that in the chain terminating at that block $\frac{2}{3}$ commits for that hash have been processed.

This gives substantial gains in implementation simplicity, because this means that we can now have a fork choice rule where the "score" of a block only depends on the block and its children, putting it into a similar category as more traditional PoW-based fork choice rules such as the longest chain rule and GHOST. However, this fork choice rule is also *finality-bearing*: it is impossible for two incompatible checkpoints to be finalized unless at least $\frac{1}{3}$ of the validators violated a *slashing condition* (see below).

There are two slashing conditions:

1. **NO_DBL_PREPARE**: a validator cannot prepare two different checkpoints for the same epoch.

2. **PREPARE_COMMIT_CONSISTENCY**: if a validator has made a commit with epoch n, they cannot make a prepare with $epoch > n$ and $epoch_{source} < n$.

Earlier versions of Casper had four slashing conditions, but we can reduce to two because of the three modifications above; they ensure that blocks will not register commits or prepares that violate the other two conditions.

2 Proof Sketch of Safety and Plausible Liveness

We give a proof sketch of two properties of this scheme: *safety* and *plausible liveness*. Safety means that two incompatible checkpoints cannot be finalized unless at least $\frac{1}{3}$ of validators violate a slashing condition. Plausible liveness means that it is always possible for $\frac{2}{3}$ of honest validators to finalize a new checkpoint, regardless of what previous events took place.

Suppose that two incompatible checkpoints A (epoch e_A) and B (epoch e_B) are finalized:

[diagram]

This implies $\frac{2}{3}$ commits and $\frac{2}{3}$ prepares in epochs e_A and e_B. In the trivial case where $e_A = e_B$, this implies that some intersection of $\frac{1}{3}$ of validators must have violated **NO_DBL_PREPARE**. In other cases, there must exist two chains $e_A > e_A^1 > e_A^2 > ... > G$ and $e_B > e_B^1 > e_B^2 > ... > G$ of justified checkpoints, both terminating at the genesis. Suppose without loss of generality that $e_A > e_B$. Then, there must be some e_A^i that either $e_A^i = e_B$ or $e_A^i >$

$e_B > e_A^{i+1}$. In the first case, since A^i and B both have $\frac{2}{3}$ prepares, at least $\frac{1}{3}$ of validators violated **NO_DBL_PREPARE**. Otherwise, B has $\frac{2}{3}$ commits and there exist $\frac{2}{3}$ prepares with $epoch > B$ and $epoch_{source} < B$, so at least $\frac{1}{3}$ of validators violated **PREPARE_COMMIT_CONSISTENCY**. This proves safety.

Now, we prove liveness. Suppose that all existing validators have sent some sequence of prepare and commit messages. Let M with epoch e_M be the highest-epoch checkpoint that was justified. Honest validators have not committed on any block which is not justified. Hence, neither slashing condition stops them from making prepares on a child of M, using e_M as $epoch_{source}$, and then committing this child.

3 Dynamic Validator Sets

We define the following constants and functions:

- $BIR(D)$: determines the base interest rate paid to each validator, taking as an input the current total quantity of deposited ether.
- $BP(D, e, LFE)$: determines the "base penalty constant" - a value expressed as a percentage rate that is used as the "scaling factor" for all penalties; for example, if at the current time $BP(...) = 0.001$, then a penalty of size 1.5 means a validator loses 0.15% of their deposit. Takes as inputs the current total quantity of deposited ether D, the current epoch e and the last finalized epoch LFE. Note that in a "perfect" protocol execution, $e - LFE$ always equals 1.
- NCP ("non-commit penalty"): the penalty for not committing, if there was a justified hash which the validator *could* have committed
- $NCCP(\alpha)$ ("non-commit collective penalty"): if α of validators are not seen to have committed during an epoch, and that epoch had a justified hash so any validator *could* have committed, then all validators are charged a penalty proportional to $NCCP(\alpha)$. Must be monotonically increasing, and satisfy $NCCP(0) = 0$.
- NPP ("non-prepare penalty"): the penalty for not preparing

- $NPCP(\alpha)$ ("non-prepare collective penalty"): if α of validators are not seen to have prepared during an epoch, then all validators are charged a penalty proportional to $NCCP(\alpha)$. Must be monotonically increasing, and satisfy $NPCP(0) = 0$.

Note that preparing and committing does not guarantee that the validator will not incur NPP and NCP; it could be the case that either because of very high network latency or a malicious majority censorship attack, the prepares and commits are not included into the blockchain in time and so the incentivization mechanism does not know about them. For $NPCP$ and $NCCP$ similarly, the α input is the portion of validators whose prepares and commits are *included*, not the portion of validators who *tried to send* prepares and commits.

When we talk about preparing and committing the "correct value", we are referring to the $hash$ and $epoch_{source}$ and $hash_{source}$ recommended by the protocol state, as described above.

We now define the following reward and penalty schedule, which runs every epoch.

- Let D be the current total quantity of deposited ether, and $e - LFE$ be the number of epochs since the last finalized epoch.

- All validators get a reward of $BIR(D)$ every epoch (eg. if $BIR(D) = 0.0002$ then a validator with 10000 coins deposited gets a per-epoch reward of 2 coins)

- If the protocol does not see a prepare from a given validator during the given epoch, they are penalized $BP(D, e, LFE) * NPP$

- If the protocol saw prepares from portion p_p validators during the given epoch, *every* validator is penalized $BP(D, e, LFE) * NPCP(1 - p_p)$

- If the protocol does not see a commit from a given validator during the given epoch, and a prepare was justified so a commit *could have* been seen, they are penalized $BP(D, E, LFE) * NCP$.

- If the protocol saw commits from portion p_c validators during the given epoch, and a prepare was justified so any validator *could have* committed, then *every* validator is penalized $BP(D, e, LFE) * NCCP(1 - p_p)$

This is the entirety of the incentivization structure, though without functions and constants defined; we will define these later, attempting as much as possible to derive the specific values from desired objectives and first principles. For now we will only say that all constants are positive and all functions output non-negative values for any input within their range. Additionally, $NPCP(0) = NCCP(0) = 0$ and $NPCP$ and $NCCP$ must both be nondecreasing.

4 Claims

We seek to prove the following:

- If each validator has less than $\frac{1}{3}$ of total deposits, then preparing and committing the value suggested by the proposal mechanism is a Nash equilibrium.
- Even if all validators collude, the ratio between the harm incurred by the protocol and the penalties paid by validators is bounded above by some constant. Note that this requires a measure of "harm incurred by the protocol"; we will discuss this in more detail later.
- The *griefing factor*, the ratio between penalties incurred by validators who are victims of an attack and penalties incurred by the validators that carried out the attack, can be bounded above by 2, even in the case where the attacker holds a majority of the total deposits.

5 Individual choice analysis

The individual choice analysis is simple. Suppose that the proposal mechanism selects a hash H to prepare for epoch e, and the Casper incentivization mechanism specifies some $epoch_{source}$ and $hash_{source}$. Because, as per definition of the Nash equilibrium, we are assuming that all validators except for one particular validator that we are analyzing is following the equilibrium strategy, we know that $\geq \frac{2}{3}$ of validators prepared in the last epoch and so $epoch_{source} = e - 1$, and $hash_{source}$ is the direct parent of H.

Hence, the PREPARE_COMMIT_CONSISTENCY slashing condition poses no barrier to preparing $(e, H, epoch_{source}, hash_{source})$. Since, in epoch e, we are assuming that all other validators *will* prepare these values and then commit H, we know H will be the hash in the main chain, and so a validator will pay a penalty proportional to NPP (plus a further penalty from their marginal contribution to the $NPCP$ penalty) if they do not prepare $(e, H, epoch_{source}, hash_{source})$, and they can avoid this penalty if they do prepare these values.

We are assuming that there are $\frac{2}{3}$ prepares for $(e, H, epoch_{source}, hash_{source})$, and so PREPARE_REQ poses no barrier to committing H. Committing H allows a validator to avoid NCP (as well as their marginal contribution to $NCCP$). Hence, there is an economic incentive to commit H. This shows that, if the proposal mechanism succeeds at presenting to validators a single primary choice, preparing and committing the value selected by the proposal mechanism is a Nash equilibrium.

6 Collective choice model

To model the protocol in a collective-choice context, we will first define a *protocol utility function*. The protocol utility function defines "how well the protocol execution is doing". The protocol utility function cannot be derived mathematically; it can only be conceived and justified intuitively.

Our protocol utility function is:

$$U = \sum_{e=0}^{e_c} -log_2(e - LFE(e)) - M * F$$

Where:

- e is the current epoch, going from epoch 0 to e_c, the current epoch
- $LFE(e)$ is the last finalized epoch before e
- M is a very large constant
- F is 1 if a safety failure has taken place, otherwise 0

The second term in the function is easy to justify: safety failures are very bad. The first term is trickier. To see how the first term works, consider the case where every epoch such that e mod N, for some N, is zero is finalized and other epochs are not. The average total over each N-epoch slice will be roughly $\sum_{i=1}^{N} -log_2(i) \approx N * (log_2(N) - \frac{1}{ln(2)})$. Hence, the utility per block will be roughly $-log_2(N)$. This basically states that a blockchain with some finality time N has utility roughly $-log(N)$, or in other words *increasing the finality time of a blockchain by a constant factor causes a constant loss of utility.* The utility difference between 1 minute finality and 2 minute finality is the same as the utility difference between 1 hour finality and 2 hour finality.

This can be justified in two ways. First, one can intuitively argue that a user's psychological estimation of the discomfort of waiting for finality roughly matches this kind of logarithmic utility schedule. At the very least, it should be clear that the difference between 3600 second finality and 3610 second finality feels much more negligible than the difference between 1 second finality and 11 second finality, and so the claim that the difference between 10 second finality and 20 second finality is similar to the difference between 1 hour finality and 2 hour finality should not seem farfetched. Second, one can look at various blockchain use cases, and see that they are roughly logarithmically uniformly distributed along the range of finality times between around 200 miliseconds ("Starcraft on the blockchain") and one week (land registries and the like).

Now, we need to show that, for any given total deposit size, $\frac{loss\_to\_protocol\_utility}{validator\_penalties}$ is bounded. There are two ways to reduce protocol utility: (i) cause a safety failure, and (ii) have $\geq \frac{1}{3}$ of validators not prepare or not commit to prevent finality. In the first case, validators lose a large amount of deposits for violating the slashing conditions. In the second case, in a chain that has not been finalized for $e - LFE$ epochs, the penalty to attackers is

$$min(NPP * \frac{1}{3} + NPCP(\frac{1}{3}), NCP * \frac{1}{3} + NCCP(\frac{1}{3})) * BP(D, e, LFE)$$

To enforce a ratio between validator losses and loss to protocol utility, we set:

$$BP(D, e, LFE) = \frac{k}{D^p} + k_2 * floor(log_2(e - LFE))$$

The first term serves to take profits for non-committers away; the second term creates a penalty which is proportional to the loss in protocol utility.

7 Griefing factor analysis

Griefing factor analysis is important because it provides one way to quanitfy the risk to honest validators. In general, if all validators are honest, and if network latency stays below the length of an epoch, then validators face zero risk beyond the usual risks of losing or accidentally divulging access to their private keys. In the case where malicious validators exist, however, they can interfere in the protocol in ways that cause harm to both themselves and honest validators.

We can approximately define the "griefing factor" as follows:

Definition 1 *A strategy used by a coalition in a given mechanism exhibits a griefing factor B if it can be shown that this strategy imposes a loss of $B * x$ to those outside the coalition at the cost of a loss of x to those inside the coalition. If all strategies that cause deviations from some given baseline state exhibit griefing factors less than or equal to some bound B, then we call B a griefing factor bound.*

A strategy that imposes a loss to outsiders either at no cost to a coalition, or to the benefit of a coalition, is said to have a griefing factor of infinity. Proof of work blockchains have a griefing factor bound of infinity because a 51% coalition can double its revenue by refusing to include blocks from other participants and waiting for difficulty adjustment to reduce the difficulty. With selfish mining, the griefing factor may be infinity for coalitions of size as low as 23.21%.

Let us start off our griefing analysis by not taking into account validator churn, so the validator set is always the same. In Casper, we can identify the following deviating strategies:

1. A minority of validators do not prepare, or prepare incorrect values.
2. (Mirror image of 1) A censorship attack where a majority of validators does not accept prepares from a minority of validators (or other isomorphic attacks such as waiting for the minority to prepare hash $H1$ and then preparing $H2$, making $H2$ the dominant chain and denying the victims their rewards)
3. A minority of validators do not commit.

4. (Mirror image of 3) A censorship attack where a majority of validators does not accept commits from a minority of validators

Notice that, from the point of view of griefing factor analysis, it is immaterial whether or not any hash in a given epoch was justified or finalized. The Casper mechanism only pays attention to finalization in order to calculate $DF(D, e, LFE)$, the penalty scaling factor. This value scales penalties evenly for all participants, so it does not affect griefing factors.

Let us now analyze the attack types:

Attack	Amount lost by attacker	Amount lost by victims
Minority of size $\alpha < \frac{1}{2}$ non-prepares	$NPP * \alpha + NPCP(\alpha) * \alpha$	$NPCP(\alpha) * (1 - \alpha)$
Majority censors $\alpha < \frac{1}{2}$ prepares	$NPCP(\alpha) * (1 - \alpha)$	$NPP * \alpha + NPCP(\alpha) * \alpha$
Minority of size $\alpha < \frac{1}{2}$ non-commits	$NCP * \alpha + NCCP(\alpha) * \alpha$	$NCCP(\alpha) * (1 - \alpha)$
Majority censors $\alpha < \frac{1}{2}$ commits	$NCCP(\alpha) * (1 - \alpha)$	$NCP * \alpha + NCCP(\alpha) * \alpha$

In general, we see a perfect symmetry between the non-commit case and the non-prepare case, so we can assume $\frac{NCCP(\alpha)}{NCP} = \frac{NPCP(\alpha)}{NPP}$. Also, from a protocol utility standpoint, we can make the observation that seeing $\frac{1}{3} \leq p_c < \frac{2}{3}$ commits is better than seeing fewer commits, as it gives at least some economic security against finality reversions, so we do want to reward this scenario more than the scenario where we get $\frac{1}{3} \leq p_c < \frac{2}{3}$ prepares. Another way to view the situation is to observe that $\frac{1}{3}$ non-prepares causes *everyone* to non-commit, so it should be treated with equal severity.

In the normal case, anything less than $\frac{1}{3}$ commits provides no economic security, so we can treat $p_c < \frac{1}{3}$ commits as equivalent to no commits; this thus suggests $NPP = 2 * NCP$. We can also normalize $NCP = 1$.

Now, let us analyze the griefing factors, to try to determine an optimal shape for $NCCP$. The griefing factor for non-committing is:

$$\frac{(1 - \alpha) * NCCP(\alpha)}{\alpha * (1 + NCCP(\alpha))}$$

The griefing factor for censoring is the inverse of this. If we want the griefing factor for non-committing to equal one, then we could compute:

$$\alpha * (1 + NCCP(\alpha)) = (1 - \alpha) * NCCP(\alpha)$$

$$\frac{1 + NCCP(\alpha)}{NCCP(\alpha)} = \frac{1 - \alpha}{\alpha}$$

$$\frac{1}{NCCP(\alpha)} = \frac{1 - \alpha}{\alpha} - 1$$

$$NCCP(\alpha) = \frac{\alpha}{1 - 2\alpha}$$

Note that for $\alpha = \frac{1}{2}$, this would set the $NCCP$ to infinity. Hence, with this design a griefing factor of 1 is infeasible. We *can* achieve that effect in a different way - by making NCP itself a function of α; in this case, $NCCP = 1$ and $NCP = max(0, 1 - 2 * \alpha)$ would achieve the desired effect. If we want to keep the formula for NCP constant, and the formula for $NCCP$ reasonably simple and bounded, then one alternative is to set $NCCP(\alpha) = \frac{\alpha}{1-\alpha}$; this keeps griefing factors bounded between $\frac{1}{2}$ and 2.

8 Pools

In a traditional (ie. not sharded or otherwise scalable) blockchain, there is a limit to the number of validators that can be supported, because each validator imposes a substantial amount of overhead on the system. If we accept a maximum overhead of two consensus messages per second, and an epoch time of 1400 seconds, then this means that the system can handle 1400 validators (not 2800 because we need to count prepares and commits). Given that the number of individual users interested in staking will likely exceed 1400, this necessarily means that most users will participate through some kind of "stake pool".

Two other reasons to participate in stake pools are (i) to mitigate *key theft risk* (i.e. an attacker hacking into their online machine and stealing the key), and (ii) to mitigate *liveness risk*, the possibility that the validator node will go offline, perhaps because the operator does not have the time to manage a high-uptime setup.

There are several possible kinds of stake pools:

- **Fully centrally managed**: users $B_1 ... B_n$ send coins to pool operator A. A makes a few deposit transactions containing their combined balances, fully controls the prepare and commit process, and occasionally withdraws one of their deposits to accommodate users wishing to withdraw their balances. Requires complete trust.

- **Centrally managed but trust-reduced**: users $B_1 ... B_n$ send coins to a pool contract. The contract sends a few deposit transactions containing their combined balances, assigning pool operator A control over the prepare and commit process, and the task of keeping track of withdrawal requests. A occasionally withdraws one of their deposits to accommodate users wishing to withdraw their balances; the withdrawals go directly into the contract, which ensures each user's right to withdraw a proportional share. Users need to trust the operator not to get their coins lose, but the operator cannot steal the coins. The trust requirement can be reduced further if the pool operator themselves contributes a large portion of the coins, as this will disincentivize them from staking maliciously.

- **2-of-3**: a user makes a deposit transaction and specifies as validation code a 2-of-3 multisig, consisting of (i) the user's online key, (ii) the pool operator's online key, and (iii) the user's offline backup key. The need for two keys to sign off on a prepare, commit or withdraw minimizes key theft risk, and a liveness failure on the pool side can be handled by the user sending their backup key to another pool.

- **Multisig managed**: users $B_1 ... B_n$ send coins to a pool contract that works in the exact same way as a centrally managed pool, except that a multisig of several semi-trusted parties needs to approve each prepare and commit message.

- **Collective**: users $B_1 ... B_n$ send coins to a pool contract that that works in the exact same way as a centrally managed poolg , except that a threshold signature of at least portion p of the users themselves (say, $p = 0.6$) needs to approve each prepare and commit messagge.

We expect pools of different types to emerge to accomodate smaller users. In the long term, techniques such as blockchain sharding will make it possible to increase the number of users that can participate as validators directly, and techniques that allow validators to temporarily "log out" of the validator set when they are offline can mitigate liveness risk.

9 Conclusions

The above analysis gives a parametrized scheme for incentivizing in Casper, and shows that it is a Nash equilibrium in an uncoordinated-choice model with a wide variety of settings. We then attempt to derive one possible set of specific values for the various parameters by starting from desired objectives, and choosing values that best meet the desired objectives. This analysis does not include non-economic attacks, as those are covered by other materials, and does not cover more advanced economic attacks, including extortion and discouragement attacks. We hope to see more research in these areas, as well as in the abstract theory of what considerations should be taken into account when designing reward and penalty schedules.

10 References

Optimal selfish mining strategies in Bitcoin; Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar: https://arxiv.org/pdf/1507.06183.pdf