Incentives in Casper the Friendly Finality Gadget

Vitalik Buterin
Ethereum Foundation

August 27, 2017

Abstract

We give an introduction to the incentives in the Casper the Friendly Finality Gadget protocol, and show how the protocol behaves under individual choice analysis, collective choice analysis and griefing factor analysis. We show that (i) the protocol is a Nash equilibrium assuming any individual validator's deposit makes up less than $\frac{1}{3}$ of the total, (ii) collectively, the validators lose from causing protocol faults, and there is a minimum ratio between the losses incurred by the validators and the seriousness of the fault, and (iii) the griefing factor can be bounded above by 1, though we will prefer an alternative model that bounds the griefing factor at 2 in exchange for other benefits. We also describe tradeoffs between protocol fairness and incentivization and fallbacks to extra-protocol resolution mechanisms such as market-driven chain splits.

We assume the "Casper the Friendly Finality Gadget" paper as a dependency.

1 Recap: The Casper Protocol

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

$$\langle \textbf{prepare}, \mathrm{h}, e, \mathrm{h}_\star, e_\star, \mathcal{S} \rangle$$

Notation	**Description**
h	a checkpoint hash
e	the epoch of the checkpoint
$\mathrm{h}_\star$	the most recent justified hash
$e_\star$	the epoch of $\mathrm{h}_\star$
$\mathcal{S}$	signature of $(\mathrm{h}, e, \mathrm{h}_\star, e_\star)$ from the validator's private key

$$\langle \textbf{commit}, \mathrm{h}, e, \mathcal{S} \rangle$$

Notation	**Description**
h	a checkpoint hash
e	the epoch of the checkpoint
$\mathcal{S}$	signature from the validator's private key

The blockchain state maintains the *current validator set* $V_c : v \rightarrow R^+$, a mapping of validators to their deposit sizes (non-negative real numbers) and the *previous validator set* $V_p : v \rightarrow R^+$. The *total current deposit size* is equal to $\sum_{v \in V} V_c[v]$, the sum of all deposits in the current validator set, and the *total previous deposit size* is likewise equal to the $\sum_{v \in V} V_p[v]$. Validators can deposit n coins to join both validator sets with deposit size n, and a validator with deposit size n' can withdraw n' coins with a delay. For any deposit or withdraw action to fully take effect, three checkpoints need to be finalized in a chain after the withdraw is included in that chain (validators get inducted to and ejected from the current validator set first, so after two finalized hashes, a validator will be in one validator set but not the other).

An *epoch* is a range of 100 blocks (e.g. blocks 600...699 are epoch 6), and a *checkpoint* as the hash of a block right before the start of an epoch. The *epoch of a checkpoint* is the epoch *after* the checkpoint, e.g. the epoch of a checkpoint which is the hash of some block 599 is 6.

We will use "p of validators" for any fraction p (eg. $\frac{2}{3}$) as shorthand for "some set of validators V_s such that $\sum_{v \in V_s} V_c[v] \geq \sum_{v \in V_c} V_c[v] * p$ and $\sum_{v \in V_s} V_p[v] \geq \sum_{v \in V_p} V_p[v] * p$" - that is, such a set must make up *both* p of the current validator set *and* p of the previous validator set. The "portion of validators that did X" refers to the largest value $0 \leq p \leq 1$ such that p of validators (using the definition above) did X.

Every checkpoint hash h has one of three possible states: *fresh*, *justified*, and *finalized*. Every hash starts as *fresh*. A hash h converts from fresh to *justified* if $\frac{2}{3}$ of validators send prepares for h with the same $(e, \mathrm{h}_\star, e_\star)$ triplet. An "ideal execution" of the protocol is one where, at the start of every epoch, every validator prepares and commits the same checkpoint for that epoch, specifying the same $e_\star$ and $\mathrm{h}_\star$; thus, in every epoch, that checkpoint gets finalized. We wish to incentivize this ideal execution.

Possible deviations from this ideal execution that we want to minimize or avoid include:

- Safety failures, i.e. two incompatible checkpoints getting finalized.
- Liveness failures, i.e. a checkpoint not getting finalized during some epoch.

These are both failures *of the protocol*. The next step from here is *fault assignment* - if a failure of the protocol were to happen, determine what failures *of individual validators* could have caused it, so that we can penalize them.

1.1 Safety faults

There exists a proof that any safety fault can only be caused by at least $\frac{1}{3}$ of validators violating one of the two Casper Commandments ("slashing conditions"), defined below:

I. A VALIDATOR SHALT NOT PUBLISH TWO OR MORE NONIDENTICAL PREPARES FOR SAME EPOCH.

In other words, a validator may Prepare at most exactly one (h , $e_\star$, $\mathrm{h}_\star$) triplet for any given epoch e .

II. A VALIDATOR SHALT NOT PUBLISH AN COMMIT BETWEEN THE EPOCHS OF A PREPARE STATEMENT.

Equivalently, a validator may not publish

$$\langle \mathbf{prepare}, e_p, \mathrm{h}_p, e_\star, \mathrm{h}_\star, \mathcal{S} \rangle \qquad \text{AND} \qquad \langle \mathbf{commit}, e_c, \mathrm{h}_c, \mathcal{S} \rangle \ , \quad (1)$$

where the epochs satisfy $e_\star < e_c < e_p$.

Hence, we can adequately penalize safety failures by simply taking away the deposits of any validator that violates either of the two slashing conditions.

1.2 Liveness Faults

Penalizing liveness faults is more difficult. If the only kind of faulty behavior that were possible is nodes going offline, then penalization would also be simple: find the validators that did not send prepares and commits during any epoch, and take away their deposits. However, there are several other faulty behaviors that are possible:

1. Preparing or committing too late
2. Preparing a different h from the hash prepared by most other validators.
3. Using a different $h_\star$ and $e_\star$ from that used by most other validators.
4. Network latency
5. A majority coalition finalizing a chain that does not include prepares or commits sent by those outside of some coalition (a “censorship fault”)
6. A majority coalition waiting for other validators to prepare one h , and then preparing another h instead.
7. A majority coalition waiting for other validators to prepare with one $h_\star$, and then preparing another $h_\star$ instead.

The list above is deliberately organized symmetrically, to illustrate a fundamental problem with attributing liveness faults known as *speaker/listener fault equivalence*: given only a transcript of messages that were sent earlier, that contain a record of user B sending a message that shows the absence of an expected message from user A, this could arise because A was not speaking, or because B was not listening, and *there is no way to tell the two apart*. In this case, (1) and (5) are indistinguishable, as are (2) and (6), and (3) and (7). Finally, all seven may be indistuiguishable from network latency.

What this means is that, in a liveness fault, we cannot unambiguously determine who was at fault, and this creates a fundamental tension between *disincentivizing harm* and *fairness* - between sufficiently penalizing validators who are malicious and not excessively penalizing validators who are not at fault. A protocol which absolutely ensures that innocent validators will not lose money must thus rely only on rewards, not on penalties, for discouraging non-uniquely-attributable faults, and so will only have a cryptoeconomic security margin equal to the size of the rewards that it issues. A protocol that penalizes suspected validators to the maximum will be one where innocent validators will not feel comfortable participating, which itself reduces security.

A third "way out" is punting to off-chain governance. If a fault could have been caused by either A or B, then split the chain in half, on one branch penalize A, on the other branch penalize B, and let the market sort it out. We can theorize that the market will prefer branches where malfeasant validators control a smaller portion of the validator set, and so on the chain that "wins" the validators that the market subjectively deems to have been responsible for the fault will lose money and the innocent valdidators will not.

[diagram]

However, there must be some cost to triggering a "governance event"; otherwise, attackers could trigger these events as a deliberate strategy in order to breed continual chaos among the users of a blockchain. The social value of blockchains largely comes from the fact that their progression is mostly automated, and so the more we can reduce the need for users to appeal to the social layer the better.

2 Rewards and Penalties

We define the following nonnegative functions, all of which return a nonnegative scalar with no units. Technically these values can exceed 1.0; in any situation which appears to call for reducing some validator's deposit size to a negative value, the deposit size should instead simply be reduced to zero.

- BIR(TD): returns the base interest rate paid to a validator, taking as an input the current total quantity of deposited coins.

- $\text{BP}(\text{TD}, e - e_{\text{LF}})$: returns 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 $\text{BP}(\cdot,\cdot,\cdot) = 0.001$, then a penalty of 1.5 means a validator loses 0.15% of their deposit. Takes as inputs the current total quantity of deposited coins TD, the current epoch e and the last finalized epoch e_{LF} . Note that in a perfect protocol execution, $e - e_{\text{LF}} = 1$.

- NPCP(α) ("non-prepare collective penalty"): if α of validators ($0 \leq \alpha \leq 1$) are not seen to have Prepared during an epoch, then *all* validators are charged a penalty of NCCP(α). NPCP must be monotonically increasing, and satisfy NPCP(0) = 0.

- NCCP(α) ("non-commit collective penalty"): if α of validators ($0 \leq \alpha \leq 1$) 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(α). NCCP must be monotonically increasing, and satisfy NCCP(0) = 0.

We also define the following nonnegative constants:

- NPP ("non-prepare penalty"): the penalty for not Preparing any block during the epoch. correct?

- NCP ("non-commit penalty"): the penalty for not Committing any block during the epoch, if there was a justified hash which the validator *could* have Committed. correct?

Note that a validator publishing a Prepare/Commit doesn't entail escaping a NPP /NCP ; it could be the case that either because of 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 see them. Likewise, for NPCP and NCCP, the α input is the proportion of validators whose Prepares and Commits are *not visible*, not the proportion of validators who *tried to send* a Prepare/Commit.

When we talk about Preparing and Committing the "correct value", we are referring to the hash h and the parent epoch $e_\star$ and parent hash $\text{h}_\star$.

We define the following reward and penalty schedule. This is the procedure for rewards and penalties, and is the entirety of the incentivization structure. It runs at the *end* of every epoch:

1. All validators get a reward of BIR(TD) (e.g., if BIR(TD) = 0.0002 then a validator with $10,000$ coins deposited gets a per-epoch reward of 2 coins)

2. If the protocol does not see a Prepare from a given validator during the epoch, the validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NPP}$ how does the incentive mechanism know e ?

3. If the protocol does not see a Commit from a given validator during the epoch, and a block was justified (so a Commit *could have* been seen), the validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NCP}$.

4. If the protocol saw Prepares from proportion p validators during the epoch, then *every* validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NPCP}(1 - p)$.

5. If the protocol saw Commits from proportion p validators during the epoch, and a block was justified (so validators *could have* Commited), then *every* validator is penalized $\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) * \mathrm{NCCP}(1 - p)$.

6. The blockchain's recorded e_{LF} and h_{LF} are updated to the latest values. correct?

3 Three theorems

We seek to prove the following:

Theorem 1 (First theorem). *If no validator has more than $\frac{1}{3}$ of the total deposit, i.e., $\max_i(D) \leq \frac{\mathrm{TD}}{3}$, then Preparing the last blockhash of the previous epoch and then Committing that hash is a Nash equilibrium. (Section 3.1)*

Theorem 2 (Second theorem). *Even if all validators collude, the ratio of the harm inflicted on the network and the penalties paid by the colluding validators is upperbounded by some constant. (Section 3.2) Note that this requires a measure of "harm inflicted".*

Theorem 3 (Third theorem). *Even when the attackers hold a majority of the total deposit, the ratio of the penalty incurred by the victims of an attack and penalty incurred by the attackers, or* griefing factor*, is at most 2. (Section 3.3)*

3.1 Individual choice analysis

The individual choice analysis is simple. Suppose that during epoch e the proposal mechanism Prepares a hash h and the Casper incentivization mechanism specifies some $e_\star$ and $\mathrm{h}_\star$. Because, as per definition of the Nash equilibrium, we are assuming that all validators except for the validator that we are analyzing are following the equilibrium strategy, we know that $\geq \frac{2}{3}$ of validators Prepared in the last epoch and so $e_\star = e - 1$, and $\mathrm{h}_\star$ is the direct parent of h.

Hence, the PREPARE_COMMIT_CONSISTENCY slashing condition poses no barrier to Preparing $(e, \mathrm{h}, e_\star, \mathrm{h}_\star)$. Since, in epoch e , we are assuming that all other validators *will* Prepare these values and then Commit h , we know h will be a hash in the main chain, and so a validator will pay a penalty if they do not Prepare $(e, \mathrm{h}, e_\star, \mathrm{h}_\star)$, and they can avoid the penalty if they do Prepare these values.

We are assuming there are $\frac{2}{3}$ Prepares for $(e, \mathrm{h}, e_\star, \mathrm{h}_\star)$, and so PREPARE_REQ also poses no barrier to committing h . Committing h allows a validator to avoid NCP . 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.

	Action	**Payoff**
Preparing $\langle e, \mathrm{h}, e_\star, \mathrm{h}_\star \rangle$	Preparing	0
	Not Preparing	$-\mathrm{NPP} - \mathrm{NPCP}(\alpha)$
	Action	**Payoff**
Committing $\langle e, \mathrm{h} \rangle$	Commiting	0
	Not Commiting	$-\mathrm{NCP} - \mathrm{NCCP}(\alpha)$

Table 1: Payoffs for ideal individual behaviors.

3.2 Collective choice model

To model the protocol in a collective-choice context, we first define a *protocol utility function*. The protocol utility function quantifies "how well the protocol execution is doing". Although our specific protocol utility function cannot be derived from first principles, we can intuitively justify it. We define our protocol utility function as,

$$U \equiv \sum_{k=0}^{e} -\log_2 \left[k - e_{\text{LF}}\right] - MF \ . \qquad (2)$$

the above equation might be able to simplifiable

Where:

- e is the current epoch, starting from 0.
- e_{LF} is the index of the last finalized epoch. To be clear, does the e_{LF} change with the term k, or is it fixed?
- M is a very large constant.
- F is an Indicator Function. It returns 1 if a safety failure has taken place, otherwise 0. A safety failure is defined as the mechanism finalizing two conflicting blocks. This is discussed in Apppendix A

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 $\sum_{i=1}^{N} -\log_2(i) \approx N * \left[\log_2(N) - \frac{1}{\ln(2)}\right]$. 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 and 2 minute finality is the same as the utility difference between 1 hour and 2 hour finality.

This can be justified in two ways. First, one can intuitively argue that a user's psychological discomfort of waiting for finality roughly matches a logarithmic schedule. At the very least, the difference between 3600 sec and 3610 sec finality feels much more negligible than the difference between 1 sec and 11 sec finality, and so the claim that the difference between 10 sec and 20 sec finality is similar to the difference between 1 hour finality and 2 hour finality seems reasonable.[1]

[1]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). add a citation for this or delete.

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, or (ii) prevent finality by having $> \frac{1}{3}$ of deposit-weighted validators not Prepare or Commit to the same hash. Causing a safety failure requires violating one of the Casper Commandments (Section 1) and thus ensures immense loss in deposits. In the second case, in a chain that has not been finalized for $e - e_{\mathrm{LF}}$ epochs, the penalty to attackers is at least,

$$\min\left[\mathrm{NPP}\left(\frac{1}{3}\right) + \mathrm{NPCP}\left(\frac{1}{3}\right), \mathrm{NCP}\left(\frac{1}{3}\right) + \mathrm{NCCP}\left(\frac{1}{3}\right)\right] * \mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) \left(\frac{1}{3}\right) \min\left[\mathrm{NPP} + \mathrm{NP} \tag{3}$$

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

$$\mathrm{BP}(\mathrm{TD}, e - e_{\mathrm{LF}}) \equiv \frac{k_1}{\mathrm{TD}^p} + k_2 * \lfloor \log_2(e - e_{\mathrm{LF}}) \rfloor \,. \tag{4}$$

what is p in the in the above equation?

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.

This connection between validator losses and loss to protocol utility has several consequences. First, it establishes that harming the protocolexecution is always a net loss, with the net loss increasing with the harm inflicted. Second, it establishes that the protocol approximates the properties of a *game* [**?**]. Potential games have the property that Nash equilibria of the game correspond to local maxima of the potential function (in this case, protocol utility), and so correctly following the protocol is a Nash equilibrium even in cases where attackers control $> \frac{1}{3}$ of the total deposit.

Here, the protocol utility function is not a perfect potential function, as it does not always take into account changes in the *quantity* of Prepares and Commits whereas validator rewards do, but it does come close. Could someone do better than our eq. 2?

3.3 Griefing factor analysis

Griefing factor analysis quanifies the risk to honest validators. In general, if all validators are honest, and if network latency stays below half half, right? the time of an epoch, then they face zero penalties. In the case where malicious validators exist, however, they can create penalties for themselves as well as honest validators.

We define the degree that malicious validators can create penalties for honest validators relative to their own penalties as the "griefing factor" of a game. We define this as,

$$GF(Ꭰ, C) \equiv \max_{S \in strategies(T \setminus C)} \frac{loss(C)}{\min[0, loss(Players \setminus C)]} \; . \tag{5}$$

I need to work on this equation more. I don't like it yet.

Definition 1. *A strategy used by a coalition in a given mechanism has 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 have griefing factors less than or equal to some bound B, then we call B a* griefing factor bound*. I plan to write this in terms of classical game theory.*

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%. [?]

Then to define the griefing factor over the entire game, we sum the area under the curve in Figure 2 leading to,

$$GF(Ꭰ) \equiv \int_0^1 GF(Ꭰ, \alpha)\, d\alpha \; . \tag{6}$$

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 H_1

Figure 1: Plotting the griefing factor as a function of the proportion of players coordinating to grief.

and then preparing H_2, making H_2 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 $\mathrm{BP}(D, e - e_{\mathrm{LF}})$, 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:

3.4 Shape of the penalities

There is a symmetry between the non-Prepare case and the non-Commit case, so we assume $\frac{\mathrm{NCCP}(\alpha)}{\mathrm{NCP}} = \frac{\mathrm{NPCP}(\alpha)}{\mathrm{NPP}}$. Also, from a protocol utility standpoint (3.4), increasing Commits are always useful as long as $p > \frac{1}{3}$, as it gives at least some economic security against finality reversions. However, Prepares $< \frac{2}{3}$ is exceedingly harmful as is it prevents *any* Commits.

Attack	**Amount lost by malicious validators**	**Amount lost**
Minority of size $\alpha < \frac{1}{2}$ non-Prepares	$\text{NPP} * \alpha + \text{NPCP}(\alpha) * \alpha$	$\text{NPCP}(\alpha) * ($
Majority censors $\alpha < \frac{1}{2}$ Prepares	$\text{NPCP}(\alpha) * (1 - \alpha)$	$\text{NPP} * \alpha + \text{N}$
Minority of size $\alpha < \frac{1}{2}$ non-Commits	$\text{NCP} * \alpha + \text{NCCP}(\alpha) * \alpha$	$\text{NCCP}(\alpha) * ($
Majority censors $\alpha < \frac{1}{2}$ Commits	$\text{NCCP}(\alpha) * (1 - \alpha)$	$\text{NCP} * \alpha + \text{N}$

Table 2: Attacks on the protocols and their costs to malicious validators and honest validators.

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 $\text{NPP} = 2 * \text{NCP}$. We can also normalize $\text{NCP} = 1$.

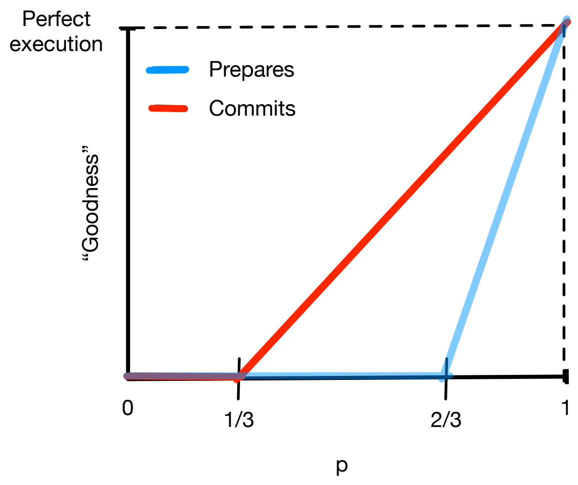
Now, let us analyze the griefing factors, to try to determine an optimal shape for NCCP. The griefing factor for non-Committing is,

$$GF = \frac{(1-\alpha) * \text{NCCP}(\alpha)}{\alpha * (1 + \text{NCCP}(\alpha))} \,. \tag{7}$$

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:

$$\begin{aligned}
\alpha * (1 + \text{NCCP}(\alpha)) &= (1-\alpha) * \text{NCCP}(\alpha) && (8)\\
\frac{1 + \text{NCCP}(\alpha)}{\text{NCCP}(\alpha)} &= \frac{1-\alpha}{\alpha} && (9)\\
\frac{1}{\text{NCCP}(\alpha)} &= \frac{1-\alpha}{\alpha} - 1 && (10)\\
\text{NCCP}(\alpha) &= \frac{\alpha}{1-2\alpha} && (11)
\end{aligned}$$

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, $\text{NCCP} = 1$ and $\text{NCP} = \max[0, 1 - 2\alpha]$ would achieve the desired effect. If we want to



Utility with function of p

NCCP and NPCP as a function of α

Figure 2: Plotting the griefing factor as a function of the proportion of players coordinating to grief.

keep the formula for NCP constant, and the formula for NCCP reasonably simple and bounded, then one alternative is to set NCCP(α) $= \frac{\alpha}{1-\alpha}$; this keeps griefing factors bounded between $\frac{1}{2}$ and 2.

4 Pools

In a traditional (i.e., 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".

There are several possible kinds of stake pools:

- **Fully centrally managed**: users $B_1 \ldots B_n$ 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 $B_1 \ldots B_n$ 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 deposits penalized, 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 using their backup key.

- **Multisig managed**: users $B_1 \dots B_n$ 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 $B_1 \dots B_n$ 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 validate directly, and extensions to allow validators to temporarily "drop out" from the validator set when they are offline can mitigate liveness risk.

5 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.

Future Work. We would like to see a better protocol utility function eq. 2. fill me in

Acknowledgements. We thank Virgil Griffith for review.

Appendix

A Safety Failure

Put the full description/definition of conflicting blocks here.

B Unused text

This is where text goes that for which a home hasn't been found yet. If no home is found, it will be deleted.

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.

Do we want to require that the Prepare be done in the first 1/2 of the epoch? I'm mildly concerned there may not always be enough time to Commit.

Remember: The only block you're allowed to Prepare is the last block of each epoch.

Remember: Even if the Finalization goes through, the collective penalties are still applied.

Questions

- It's unclear to me why we need $e_\star$ in the Prepare.