Incentives in Ethereum’s Hybrid Casper Protocol

Vitalik Buterin*, Daniël Reijsbergen†, Stefanos Leonardos†, Georgios Piliouras†

*Ethereum Foundation
†Singapore University of Technology and Design

Abstract—We present an overview of hybrid Casper the Friendly Finality Gadget (FFG): a Proof-of-Stake checkpointing protocol overlaid onto Ethereum’s Proof-of-Work blockchain. We describe its core functionalities and reward scheme, and explore its properties. Our findings indicate that Casper’s implemented incentives mechanism ensures liveness, while providing safety guarantees that improve over standard Proof-of-Work protocols. Based on a minimal-impact implementation of the protocol as a contract on the blockchain, we discuss additional issues related to parametrisation, funding, throughput and network overhead and detect potential limitations.

Index Terms—Proof of Stake, Ethereum, Consensus

I. INTRODUCTION

In 2008, the seminal Bitcoin paper by Satoshi Nakamoto [34] introduced the blockchain as a means for an open network to extend and reach consensus about a distributed ledger of digital token transfers. The main innovation of Ethereum [12] was to use the blockchain to maintain a history of code creation and execution. As such, Ethereum functions as a global computer that executes code uploaded by users in the form of smart contracts. Like Bitcoin [25], [26], Ethereum’s block proposal mechanism is based on the concept of Proof-of-Work (PoW). In PoW, network participants utilise computational power to win the right to add blocks to the blockchain. However, the ballooning global energy consumption [18], [45] of PoW-based blockchains has made the concept increasingly controversial. One of the main alternatives to PoW is virtual mining or Proof-of-Stake (PoS) [1], [2], [37]. In PoS, the right to propose a block is earned by locking – or depositing – tokens on the blockchain, which has no inherent energy cost.

As part of its long-term goal to switch from PoW to PoS, Ethereum is designing a full PoS protocol called Casper [11], [39], [40]. To ensure a smooth transition with minimal impact to its users, Ethereum deployed and tested a hybrid version, Casper the Friendly Finality Gadget or Casper FFG, as a smart contract on a dedicated testnet [27], [35], [36]. Essentially, Casper FFG is a simplified version of a Byzantine fault tolerant protocol [4], with “votes” for checkpoints taking the place of prepares and commits. In contrast to protocols that treat every block as a checkpoint (e.g., Tendermint [31] and Algorand [15]), Casper FFG periodically checkpoints blocks on an underlying chain. As such, the tried and tested PoW chain can be preserved during a transitional phase, whilst the extra load on the network is mitigated. This addresses two of the classical challenges that affect PoS protocols [4], [22]: the nothing-at-stake problem through the slashing mechanism that penalises misbehaving violators [13], and long-range attacks through a modified fork-choice rule that prioritises (and never reverts) finalised checkpoints over PoW [43].

The high-level idea and fundamental properties of the hybrid Casper FFG have been outlined in [13]. The present paper is an extension of [13] that features a full description of the implemented incentives (reward–penalty) scheme and rigorous proofs of liveness and safety for the tested version. Based on the minimal-impact implementation of Casper FFG as a smart contract on the PoW chain, the present paper also covers the parameter choice, confirmation times and network overhead, and a discussion of potential limitations.

The contributions of this paper are as follows. We first provide an overview of the Casper FFG protocol and describe its core functionalities. To reason about liveness and safety, we develop a mathematical framework for the incentives scheme, slashing conditions, and the fork-choice rule. Our first theoretical result is that with the implemented reward scheme, Casper is $\alpha$-live, for any $\alpha \in (0,1]$, i.e., online validators controlling any fraction $\alpha \in (0,1]$ of the stake will be eventually able to finalise checkpoints. Concerning safety, we first show that the property proved in [13] carries over to the updated incentives scheme, namely that two or more “conflicting” checkpoints can only exist if validators controlling at least 1/3 of the stake misbehave conspicuously and hence, can be slashed (i.e., punished). In the case of a protracted network partition, the liveness guarantee has implications for safety, as it allows for conflicting checkpoints to be finalised. However, using both numerical and analytical tools, we derive that the minimum duration of such a network partition is very large (i.e., at least three weeks). Finally, we turn to the implementation of Casper FFG as a PoW chain contract and discuss the protocol’s impact on transaction throughput, the effect of different parametrisations and potential limitations.

To remain compatible with Ethereum’s evolution towards a sharded and hence more scalable design [10], [22], [40], the specifications of Casper FFG are constantly updated [8], [11]. However, the main components, i.e., the incentives mechanism, functionality, and design philosophy remain basic components of Casper FFG even in the sharded construct (i.e., multi-chain) setting [7], [11], [20]. More broadly, since the protocol can be deployed as a smart contract on top of any chain-based blockchain, PoW or PoS [43], it can be of wider

This work was supported in part by the National Research Foundation (NRF), Prime Minister’s Office, Singapore, under its National Cybersecurity R&D Programme (Award No. NRF2016NCR-NCR002-028) and administered by the National Cybersecurity R&D Directorate. Georgios Piliouras acknowledges SUTD grant SRG ESD 2015 007, MOE AcRF Tier 2 Grant 2016-T2-1-170 and NRF 2018 Fellowship NRF-NRF2018-07.
interest to the blockchain community beyond Ethereum.

Outline: In Section II, we provide an abstract overview of Casper FFG and its operations in a formal mathematical model. Section III contains our main theoretical results on liveness and safety. In Section IV, we present our findings on Casper’s implementation and discuss related issues. We summarise our results in Section V.

II. The Hybrid Casper FFG Protocol

A. The PoW Chain

Ethereum functions as a global computer whose operations are replicated across a peer-to-peer network. Participants in the network are called nodes – they typically interact with the rest of the network via a software application called a client. The client interacts with the Ethereum blockchain via transactions. There are three main types of transactions:

Token transfers provide the same core functionality as Bitcoin by allowing nodes to exchange digital tokens.

Contract creations upload pieces of code, called (smart) contracts, to the blockchain. Contracts are executed using a runtime environment called the Ethereum Virtual Machine (EVM) \[1\]. Two notable high-level languages that compile into the EVM are Solidity and Vyper \[2\], which are based on the JavaScript and Python languages, respectively. Vyper is particularly relevant for this paper as it is used for the Casper contract. A typical contract will include one or more functions which can be called by the nodes.

Contract calls handle interactions with the functions of an existing contract.

In PoW, the ordering of these transactions in the blockchain is determined by a special class of nodes called miners. Miners collect and sort transactions, after which they execute these transactions in the EVM. The resulting information about the state of the complete global computer (account balances, contract variables, etc.) is then combined with the transactions and various other data into a data structure called a block. Miners compete to find a block that satisfies a condition that requires considerable computational effort. The winning miner receives a fixed amount of Ether (ETH) – Ethereum’s native currency – in the form of a mining reward. Additionally, all of the three transaction types listed above require gas, which is also paid to the winning miner as a reward for the computational effort. In particular, more computationally expensive operations in the EVM require more gas (see \[47\] for a complete specification of the gas cost for the different types of operations) – as such, gas cost is a good measure for the computational ‘cost’ of a transaction. In Section IV we will investigate the gas cost of the different functions in the Casper contract.

Formal Framework: To reason about the evolution of the PoW chain in the context of network latency and partitions, we need a formal framework. Let \( N \) denote the full set of nodes as identified by an integer denoting their index in the network, i.e., \( N \subseteq \mathbb{N} \). At each time \( t \geq 0 \), each node \( n \in N \) is aware of a set of blocks \( B \) which we denote by \( B_n^t \), i.e.,

\[
B_n^t := \{ B : \text{node} n \text{ is aware of} B \text{ at time} t \geq 0 \}.
\]

The genesis block \( g \) is the only block that all nodes are aware of at time \( 0 \), i.e., \( B_n^0 = \{ g \} \) for all \( n \in N \). Each block \( B \) can be represented uniquely as an integer \[2\] Due to network latency, eclipse attacks or other reasons, it may be the case that different nodes are aware of different sets of blocks, i.e., there may exist \( t > 0 \) and nodes \( n, m \in N \) such that \( B_n^t \neq B_m^t \). We assume that nodes cannot forget about blocks, i.e.,

\[
B_n^s \subseteq B_n^t, \text{ for any } s, t > 0 \text{ with } s > t.
\]

Each block \( B \) points to a previous block \( P(B) \) via a function \( P: \mathbb{N} \rightarrow \mathbb{N} \), cf. \[3\], with

- \( P_0(B) := B \) for any block \( B \),
- \( P_2(B) = P(P(B)), P_3(B) = P(P(P(B))) \) etc.
- \( P^k(g) := \emptyset \) for all \( k \geq 1 \), where \( g \) is the genesis block.
- \( P^k(B) \neq B \) for any block \( B \) and \( k \in \{1, 2, \ldots \} \), i.e., there are no cycles \[3\].

Based on this relationship, we define the chain \( C(B) \) of a block \( B \) as the path from \( B \) leading back to \( g \), i.e., \( C(B) := (B, P(B), \ldots, P^{k-1}(B), g) \). If \( B' \neq B \) is a block such that \( B' \in C(B) \) then \( B' \) is called an ancestor of \( B \) and \( B \) is called a child of \( B' \). If \( B' = P(B) \), then \( B \) is called a direct child of \( B' \). The length \( k \) of \( C(B) \) determines the block height \( h(B) \) of block \( B \), i.e., \( h(B) = k \) if \( P^k(B) = g \). The height of the genesis block is \( 0 \), i.e., \( h(g) = 0 \). The state of the blockchain in block \( B \) is obtained by executing all transactions in \( C(B) \) starting from the genesis block \( g \), see also \[27\].

A fork occurs whenever two different blocks \( A \) and \( B \) exist such that \( P(A) = P(B) \). At any time \( t \), each miner \( n \) needs to decide which block in \( B_n^t \) to extend. This is done using a fork-choice rule, which in its simplest form is a function \( f \) that maps a set \( B \) to a single block \( B \). The block chosen is called the head. In PoW, the fork-choice rule is to select the block \( B \) with the highest accumulated proof-of-work \[3\] and in case of ties to prefer the block seen first. Hybrid Casper FFG’s fork-choice rule is discussed in the next section.

B. Execution of Hybrid Casper FFG

Validators: In hybrid Casper FFG, some nodes assume the role of validators. Nodes can become validators by locking/staking tokens on the PoW chain, thus creating a deposit.

In the Casper contract, this is done by calling the deposit and withdraw functions. Additionally, the logout function removes a validator from the active validator set and needs to be called before withdrawing. Validators need to wait a long period \[4\] after depositing before being allowed to withdraw.

---

1The eWASM framework \[17\] may replace the EVM in the future.
2Hyperlinks: Solidity and Vyper.
Checkpoints: The central role of the validators is to vote for checkpoints \([42]\). A checkpoint is any block with block number \(i \cdot l\), where \(i \in \{0, 1, \ldots\\}\) and \(l \in \mathbb{N}\) denotes the epoch length: an epoch is defined as the contiguous sequence of blocks between two checkpoints, including the first but not the latter. Block 0 (which is also a checkpoint) denotes the genesis block. We will assume \(l = 50\) throughout this paper [43]. The epoch of a block is the index of the epoch containing that hash, e.g., the epoch of blocks 200 and 249 is 4. Because validators’ deposits change between epochs due to the rewards and penalties, we denote by \(D_{\nu,i}\) the deposit of validator \(\nu \in V\) at the beginning of epoch \(i \in \mathbb{N}\), where \(V\) denotes the set of active validators at epoch \(i\). Hence, let \(D_i := \sum_{\nu \in V} D_{\nu,i}\) denote the total stake (deposited value) at epoch \(i\).

Votes in Casper: In the Casper contract, voting is done by calling the vote function with the arguments \((\nu, t, h(t), h(s), S)\), where the entries are explained in Table I. We will say that a validator \(\nu\) who generates a vote transaction to the Casper address votes correctly or casts a valid vote if \(\nu\) includes the expected source and target epochs (checkpoints) returned to \(\nu\) by a Casper contract call and a valid signature created by \(\nu\)’s private key \([42, 43]\). In particular, the only valid target epoch is the current epoch (in the contract) and the only valid target for a vote that appears in block \(B\) is the checkpoint block with the correct height on the chain \(C(B)\). [7]

Finalisation: A checkpoint \(t\) is justified if at least \(2/3\) of validators – in terms of stake – publish a vote of the form \((\nu, t, h(t), h(s), S)\) for some checkpoint \(s\), such that \(s\) is an ancestor of \(t\) and is itself justified. A checkpoint \(s\) is finalised if it is justified and a direct child [8] of \(s\) is also justified – i.e., if \(t = P(s)\) and \(s, t\) are both justified, then \(s\) is finalised. We consider \(g\) to be both justified and finalised, to serve as the base case for the recursive definition. Two finalised checkpoints \(s, t\) are conflicting if neither is an ancestor of the other. When handling vote transactions the Casper contract only considers (includes) correct votes as defined above. Votes for the wrong targets and target epochs are considered invalid and ignored.

Fork-Choice Rule: The Hybrid Casper FFG fork-choice rule extends the PoW fork-choice rule of Section II-A. In particular, to select a block as the head of the chain, a client queries the contract to find the block\((s)\) with the highest justified epoch, prioritising the block with the highest mining PoW only in a case of tie. Clients only consider epochs in which the total deposit exceeds a given minimum threshold and never revert a known finalised checkpoint. If a chain has no justified checkpoints beyond \(g\), the fork-choice rule essentially reverts to pure PoW [43].

C. Rewards and Penalties Mechanism

According to Casper’s payments scheme, validators who vote correctly during an epoch are rewarded, and validators who do not are penalised. This is achieved by adjusting the validators’ deposits depending on their own voting behaviour, i.e., on whether they are voting or not, and on the performance of the protocol as a whole, i.e., on what total fraction of validators vote correctly and whether that is enough to justify and finalise checkpoints. Specifically,

Correct voting: If checkpoints are being finalised, i.e., if at least \(2/3\) of validators are voting correctly, then deposits of correctly voting validators increase by a positive interest rate. If checkpoints are not being finalised, deposits of correct voting validators remain the same. The interest rate depends on the total deposit and how many other validators are voting.

Non-voting: Regardless of finalisation, non-voting validators are penalised and their deposits shrink. The penalties are increasing in the proportion of non-voting validators. If epochs fail to be finalised during sustained time periods, then penalties become gradually more severe.

Conflicting/incorrect voting: Incorrect votes are ignored and validators who cast them are considered as non-voters and are not rewarded. If evidence is provided that validators cast conflicting votes, then their deposits are partially or entirely removed (slashed) depending on the severity of the violation and the overall protocol performance (proportion of “correct” voters and whether epochs are being finalised or not).

In Casper’s implementation, these adjustments are achieved via a scheme of reward factors, cf. \([1, 4]\) that properly update validator’s deposits, cf. \([43]\). This payment scheme is designed to make the protocol incentive compatible, i.e., to encourage validators to vote correctly and as often as possible and enforce the protocol’s purposes of liveness and safety, as we discuss in more detail in Section III.

The Scheme: In detail, the Casper contract adjusts validators’ deposits via the individual and the collective reward factors, \(\rho_i\) and \(C_i\) respectively, that depend on each validator’s voting behaviour and on the aggregate protocol functionality, i.e., on whether epochs are finalised or not, for every epoch \(i \in \mathbb{N}\).

Specifically, let \(\text{ESF}_i\) denote the Epochs Since Finalisation during epoch \(i\), defined as \(i - \text{height of the last finalised epoch}\). Since \(q\) is finalised, \(\text{ESF}_0 = 0\) and \(\text{ESF}_1 = 1\). For \(i \geq 2\), \text{ESF}_i always equals 2 in the ideal situation where everyone always votes (it requires two consecutive epochs to be justified for the first of them to be finalised), otherwise it is higher. Based on the total stake \(D_i\) in epoch \(i \in \mathbb{N}\), the individual reward factor \(\rho_i\), is then defined as follows

\[
\rho_i := \gamma \cdot D_i^{-\rho} + \beta \cdot (\text{ESF}_i - 2),
\]

TABLE I

Summary of vote function arguments.

| Notation | Description |
|----------|-------------|
| \(\nu\)  | validator index |
| \(t\)    | hash of the ‘target’ checkpoint |
| \(h(t)\) | height of the target checkpoint in the checkpoint tree |
| \(h(s)\) | height of the source checkpoint in the checkpoint tree |
| \(S\)    | signature of \((\nu, t, h(t), h(s))\) by the validator’s private key |
if $D_i > 0$, and $\rho_i = 0$ otherwise. Here, $\gamma, p, \beta > 0$ denote the base interest, total deposit dependence and base penalty factors, respectively.\footnote{The name of the parameters are self-explanatory. See Section IV-A for more details on their values and choice.} In the benchmark specification, $\gamma = 7 \cdot 10^{-3}$, $p = \frac{1}{2}$ and $\beta = 2 \cdot 10^{-7}$. (The parameter choice will be discussed further in Section IV.) To define the collective reward factor, let

$$1_{\nu, i} = \begin{cases} 1 & \text{if } \nu \text{ voted successfully during epoch } i, \\ 0 & \text{otherwise.} \end{cases}$$

At the end of epoch $i$, let $m_i$ denote the weighted fraction of correctly voting validators in epoch $i$, defined as $m_i := \frac{1}{2\gamma} \sum_{v \in V} 1_{\nu, i} D_{\nu, i}$. The collective reward factor at the beginning of epoch $i$, $C_i$, is defined as

$$C_i := \begin{cases} \frac{1}{2} m_{i-1} \rho_{i-1} & \text{if ESF}_i = 2, \\ 0 & \text{otherwise.} \end{cases}$$

(2)

Validators’ deposits are then updated via the scheme

$$D_{\nu, i} := \begin{cases} (1 + C_{i-1}) \frac{1}{1 + \rho_{i-1}} 1_{\nu, i-1} D_{\nu, i-1} & \text{if ESF}_i = 2, \\ 0 & \text{otherwise.} \end{cases}$$

(3)

In the Casper contract, the reward factors are updated by calling the function initialize_epoch once per epoch (the contract processes only the first call in each epoch).

**Slashing Conditions:** Finally, to account for conflicting votes, Casper introduces the following commandments or slashing conditions,\footnote{See also \cite{2, 28, 40} and \cite{46} for varying definitions of these concepts.} a validator $\nu$ who publishes two distinct votes

$$\langle \nu, t_1, h(t_1), h(s_1), S_1 \rangle \text{ and } \langle \nu, t_2, h(t_2), h(s_2), S_2 \rangle,$$

violates Casper’s slashing conditions, if

\begin{itemize}
  \item \textbf{I.} $h(t_1) = h(t_2)$: i.e., they are for same target height,
  \item \textbf{II.} $h(s_1) < h(s_2) < h(t_2) < h(t_1)$: i.e., one vote is within the ‘span’ of the second vote.
\end{itemize}

Any validator is able to call the slash function with the data for two potentially offending vote messages as its arguments. If the slash call is found to be valid, then the offending validator’s deposit is partially or entirely taken away (slashed) and the sender receives 4% of the validator’s deposit as a ‘finder’s fee’. Invalid calls to slash are ignored in the same manner as invalid votes. Calls to slash can be valid on any chain, even those that do not include the offending vote calls, because the function arguments and signatures by themselves are sufficient evidence for misbehaviour. The degree to which the validator’s deposit is shrunk depends on the total amount slashed ‘recently’ across the protocol – the reasoning is to punish more harshly when there is a higher risk that two conflicting checkpoints with the same height will be finalised.

**III. Analysis**

In this section we investigate Casper’s performance on the two integral goals of liveness and safety. To make the analysis self-contained, we start by providing the relevant definitions.\footnote{Block creation times are typically assumed to be exponentially distributed.}

We use the term $\alpha$-strong nodes (validators) for nodes who control a fraction $\alpha \in (0, 1]$ of the protocol resources (stake). Also, let $T_{\infty} \in (0, \infty) \cup \{\infty\}$ denote the random time until the finalisation of the next checkpoint by a set of $\alpha$-strong nodes. Randomness stems from block creation times in the underlying PoW chain.\footnote{Also, let $T_{\infty} \in (0, \infty) \cup \{\infty\}$ denote the random time until the finalisation of the next checkpoint by a set of $\alpha$-strong nodes.}

**Definition 1 (Liveness).** We say that a protocol II is $\alpha$-live for some $\alpha \in (0, 1]$, if a group of protocol-following, $\alpha$-strong nodes will be able to extend the blockchain (finalise checkpoints) after a finite period of time with probability 1.

**Definition 2 (Safety).** We say that a protocol II is $\alpha$-safe if the following holds: any protocol-following node who considers a checkpoint $i$ as final at some time point $\tau \geq 0$, will also consider $i$ as final at any time point $t > \tau$.

**Threat Model & Fault Types:** To study liveness and safety, we assume an adversary that controls at most 49\% of Casper’s resources (stake) for an infinite period of time. We note that Casper also has a defensive mechanism against majority 51\% attacks, namely the minority fork\footnote{Also, let $T_{\infty} \in (0, \infty) \cup \{\infty\}$ denote the random time until the finalisation of the next checkpoint by a set of $\alpha$-strong nodes.}. Here, we will focus on the standard threat model that assumes an honest majority. We assume a partially synchronous network, i.e., messages that do not arrive within the timespan of the intended epoch, are ignored. As we discuss further in Section IV-C, we do not account for collisions between miners and validators. We will show that Casper’s reward mechanism disincentivises the following two core types of faults:

**Liveness faults:** checkpoints not getting finalised during one or more consecutive epochs.

**Safety faults:** two or more conflicting checkpoints being finalised in the same or different epochs. (After all, this would either lead to a permanent fork, or to at least one node having to overturn a finalised checkpoint through a manual reset.)
During any period \( E \subset \mathbb{N} \) of consecutive epochs in which more than \( 1/3 \) of the validators (in terms of stake) are not voting correctly, checkpoints cannot be finalised. In the reward mechanism, this translates to an increase in the epochs since finalisation, ESF, which in turn affects both the individual \( \rho_i \) and the collective \( C_i \) reward factors, for \( i \in E \). This is captured by Lemma 3. Without loss of generality, we assume that a fault involving \( (1 - \alpha) \)-strong validators not following the protocol is initiated at epoch 0. All relevant notation is summarised in Table II.

### Table II: List of Symbols.

| Notation | Description |
|----------|-------------|
| \( l \) | length (in terms of blocks) of a Casper FFG epoch |
| \( V \) | current active validator set |
| \( D_{v,i} \) | deposit of validator \( v \in \mathbb{N} \) at the beginning \( i \) |
| \( D_i \) | total deposit at the beginning of epoch \( i \) |
| \( m_i \in [0,1] \) | weighted fraction of correct votes in epoch \( i \) |
| \( \rho_i \geq 0 \) | individual reward factor in epoch \( i \) |
| \( C_i \geq 0 \) | collective reward factor in epoch \( i \) |
| \( \gamma > 0 \) | base interest factor \( > 0 \) |
| \( \beta > 0 \) | base penalty factor |
| \( p > 0 \) | total deposit dependence factor (typically in \((0,1)\)) |
| \( \alpha \in (0,1] \) | stake fraction of honest validators |

**Lemma 3.** Let \( D_0 \) denote the initial stake in epoch 0 and let \( \gamma, \forall \) be \( \alpha \) and \((1 - \alpha)\)-strong validators, respectively, with \( \alpha \in (0,2/3) \). Assume that from epoch 0 onwards, only \( \alpha \)-strong validator \( \gamma \) continues to vote correctly. Then, for the consecutive epochs \( i \geq 1 \) that are not being finalised

1. the total deposits \( D_i \) are decreasing in \( i \),
2. \( D_i \) is given by \( D_i = \alpha D_0 + (1 - \alpha) D_0 \prod_{j=0}^{i-1} (1 + \rho_j)^{-1} \),
3. the individual reward factors \( \rho_i \) are increasing in \( i \).

**Proof.** By definition \( \gamma \), as long as epochs \( i \geq 1 \) are not being finalised, \( C_i = 0 \). Hence, by (1),

\[
D_{v,i} = (1 + 0) \frac{1 + \rho_{i-1}}{1 + \rho_{i-1}} D_{v,i-1} = D_{v,i-1},
\]

and similarly \( D_{\gamma,i} = (1 + \rho_{i-1})^{-1} D_{\gamma,i-1} < D_{\gamma,i-1} \). The last inequality holds because \( \rho_i > 0 \) for all \( i \geq 1 \) by definition. Since \( D_i = D_{v,i} + D_{\gamma,i} \) for all \( i \geq 1 \), this implies \( D_i = D_{v,i} + (1 + \rho_{i-1})^{-1} D_{\gamma,i-1} < D_{i-1} \), which shows (i). The expression in (ii) is obtained by repeated application of the previous recursive equalities, \( D_{v,i} = D_{v,i-1} \) and \( D_{\gamma,i} = (1 + \rho_{i-1}) D_{\gamma,i-1} \), and the assumption \( D_{\gamma,0} = \alpha D_0 \). Finally, to prove (iii), observe that ESF, \( -2 = i \), since the last finalised epoch is \( -2 \) and hence, by (1),

\[
\rho_i = \gamma D_{\gamma,i}^{-p} + \beta (\text{ESF}, -2) = \gamma D_{\gamma,i}^{-p} + \beta i.
\]

Since \( D_i \) is decreasing by (i) and \( p > 0 \), \( \rho_i \) is increasing which concludes the proof.

**Discussion of Lemma 3.** Intuitively, Lemma 3 says that as epochs are not getting finalised, Casper’s implemented incentives mechanism increases the deposits of validators who vote correctly – i.e., without violating any slashing conditions – and consistently and decreases the deposits of non-voting validators. As ESF increases, this process will continue until the voting validators will account for more than \( 2/3 \) of the total stake on the chain that they are voting on. The overall period that will be required depends on the initial proportion of their stake. From this point on, they will resume finalisation of checkpoints and hence, ESF will reset to 0, making changes in deposits much slower. This is the main argument behind Casper’s liveness which is formalised in the Theorem 4 and illustrated in Figure 2.

**Theorem 4.** The Casper contract is \( \alpha \)-live for any \( \alpha \in (0,1] \).

**Proof.** If correctly voting validators control more than \( 2/3 \) of the stake, then finalisation and hence, liveness are immediate. To treat the remaining case, assume that voting validators control \( \alpha < 2/3 \) of the stake at epoch 0 in which \((1-\alpha)\)-strong validators stop voting. In this case, finalisation will resume after epoch \( k \geq 1 \), if \( D_{\gamma,k} \geq 2/3 D_0 \) or equivalently if \( D_k \leq 2/3 \alpha D_0 \), since, by the proof of Lemma 3, \( D_{\gamma,k} = D_{\gamma,0} \). Hence, by Lemma 3(ii)

\[
\frac{3}{2} \alpha D_0 \geq \alpha D_0 + (1 - \alpha) D_0 \prod_{i=0}^{k-1} (1 + \rho_i)^{-1},
\]

which is equivalent to \( \prod_{i=0}^{k-1} (1 + \rho_i) \geq 2 (1 - \alpha)/\alpha \). This implies that finalisation will resume after epoch \( k_{\alpha} \), where \( k_{\alpha} \) is the solution to the following minimisation problem

\[
k_{\alpha} := \min \left\{ k \in \mathbb{N} : \prod_{i=0}^{k-1} (1 + \rho_i) \geq 2 (1 - \alpha)/\alpha \right\}
\]

Hence, it remains to show that the above problem has a finite solution \( k_{\alpha} \in \mathbb{N} \) for every \( \alpha \in (0,2/3) \). Since \( \rho_i = \gamma D_{\gamma,i}^{-p} + \beta i \geq \beta i \) by the proof of Lemma 3(iii), the standard inequality \( \prod_{i=0}^{k-1} (1 + \rho_i) \geq 1 + \sum_{i=0}^{k-1} \rho_i \), implies that it suffices to find a \( k \) such that \( \sum_{i=0}^{k-1} \rho_i = \beta (k+1)^2 \geq 2 (1 - \alpha)/\alpha \). Since \( \beta^2 \) is unbounded and \( \beta > 0 \) is constant, such a \( k \) exists for every \( \alpha \in (0,2/3) \).

Given the benchmark parameterisation of the contract (to be discussed further in Section IV-A), the number of epochs needed to resume finalisation is illustrated in Figure 3 for different groups of \( \alpha \)-strong voting validators. We emphasise the following values which we will use in the study of Casper’s safety properties: for \( \alpha = 0.33, 0.49, \) and 0.51, the number of epochs needed for \( \alpha \)-strong validators to resume finalisation is 3733, 2698, and 2546 respectively.

**Safety:** We now turn to the study of Casper’s safety properties. To explore the trade-off between liveness and safety, we distinguish two scenarios: in the first, we assume a unified network in which clients have a view of all active chains, and in the second, a partitioned network, in which each client has a view of only a single chain. In the first scenario, we seek to prevent the nothing-at-stake problem, which occurs by the incentive to finalise conflicting checkpoints on different chains during a fork. Casper’s mechanism ensures that in the short term, different checkpoints cannot be finalised unless at least \( 1/3 \) of validators violate one slashing condition. Intuitively, this relies on the fact that...
since the two conflicting justified checkpoints (this element exists in terms of block height – common element in the chains of (conflicting) chains. Taking into account that votes between two pairs of consecutive justified checkpoints on two different chains are identified by the protocol as invalid and are ignored, two cases may have occurred:

- two of the conflicting checkpoints have the same height: this directly violates slashing condition I.
- all conflicting checkpoints have different heights: this implies that one pair of consecutive justified checkpoints must be included within the span of two justified checkpoints on the conflicting chain, which violates slashing condition II.

This is the statement of Theorem 5 which we prove for Casper’s current parametrisation, cf. Section IV-A. We focus on the most favourable case for the adversaries, i.e., 2 competing chains, where all of the adversaries’ power is coordinated on the same chain.

**Theorem 6.** Assume that \( \alpha \geq 0.51 \) validators are honest and follow the protocol with the same input. Then,

1) if \( \frac{2}{3} \leq \alpha \leq 1 \), honest validators will immediately finalise a checkpoint on the canonical chain. For a conflicting checkpoint to be finalised on a competing chain with \( 1 - \alpha \) of validators, the partition should last at least 3733 epochs.

2) if \( 0.51 < \alpha < \frac{2}{3} \), honest validators will finalise blocks first after at most 2546 epochs and this is the canonical chain with overwhelming probability. For a conflicting block to be finalised, the partition should last at least 2698 epochs.

**Proof.** Consider a fork (network partition) initiated at time \( t = 0 \) and let \( f_t^i \triangleq (\alpha_1^i, 1 - \alpha_1^i) \) denote the distribution of validators (in terms of resources) in fork (chain) \( i = 1, 2 \) at time \( t \geq 0 \). This information can be stored in a matrix \( F^t \).

\[
F^t = \begin{pmatrix}
\alpha_1^1 & 1 - \alpha_1^1 \\
\alpha_1^2 & 1 - \alpha_1^2
\end{pmatrix}
\]

for \( t \geq 0 \), where \( \alpha_1^0 = \alpha_2^0 = \alpha \) denotes the initial stake of the validators voting in chain \( f_1 \) which we assume to be the honest group. Since, they vote only on chain \( f_1 \), \( \alpha_1^i \) will be non-decreasing and \( \alpha_2^i \) will be decreasing. The opposite holds for the stakes \((1 - \alpha_1^i)\) for \( i = 1, 2, t \geq 0 \) controlled by the malicious validators who are only voting on chain \( f_2 \).

**Case 1:** \( \frac{2}{3} \leq \alpha \leq 1 \). Checkpoints in chain \( f_1 \) are being finalised without interruption and hence, validators voting in \( f_2 \) know that they are in the canonical chain. By the reward scheme, their deposits in chain \( f_2 \) will shrink and at some point the deposits of (malicious) validators voting on \( f_2 \) will account for at least 2/3 of the stake on \( f_2 \). However, since at most \( 1 - \alpha_2^0 \leq 1/3 \) of validators are voting on fork \( f_2 \), the...
time that is required for this to happen is at least 3733 epochs under the current parametrisation, cf. Figure 3.

**Case II:** $0.51 \leq \alpha < \frac{2}{3}$. In this case, validators voting in $f_1$ will have to wait for at most 2698 epochs, cf. Figure 3, for their stake, $\alpha_1$, to account for 2/3 of the total stake in $f_1$. Validators voting in $f_2$ will need at least 2698 epochs to resume finalisation, cf. Figure 3. Since individual block creation times are exponentially distributed, the probability that finalisation in $f_2$ will precede finalisation in $f_1$ is less or equal to the probability that an Erlang random variable with parameters $n_1 = 2698, \lambda_1 = 0.49$ will be less or equal than a random variable $n_2 = 2546, \lambda_2 = 0.51$, which is negligible (Python calculation: 0E-537).

The argument in the proof of Theorem 6 is illustrated in Figure 4. After the initiation of the fork, validators who keep voting in the upper branch know that they will be able to start finalising first, since they form the majority at any point in time. Validators in the lower branch will also be able to finalise checkpoints, yet this will take considerably longer, see Figure 3. In this case, conflicting checkpoints will be finalised and clients aware of either of the checkpoints will not be willing to revert them (under Casper’s fork-choice rule). However, this requires the network to be partitioned for a period of time which is of theoretical interest only.

**IV. Implementation**

In this section, we discuss implementation specifics of the Casper contract, in particular the financial cost in terms of rewards to validators in Section IV-A its impact on transaction throughput in Section IV-B and potential issues and limitations in Section IV-C.

**A. Impact of the Parameter Choice**

Finding appropriate values for each of $\gamma$, $\beta$, and $p$ as discussed in Section II-C involves a tradeoff. A higher base interest $\gamma$ means that the protocol becomes more expensive to operate, but also more decentralised as more people will be willing to deposit. A higher base penalty factor $\beta$ means improved liveness, i.e., faster recovery from a large number of validators going offline. However, higher penalties also mean bigger losses for validators during serious network partitions. A higher deposit size dependence $p$ means that validators can make a larger profit by performing censorship or DoS attacks against other validators. However, setting it too low means that the protocol does not automatically adjust the interest rate depending on how risky potential validators perceive depositing to be (an argument also made in [16]).

As discussed in [43], the benchmark parameters were set as $\gamma = 7 \cdot 10^{-3}$, $\beta = 2 \cdot 10^{-7}$, and $p = 1/2$. Here, $p$ was chosen to strike a balance between $p = 0$ (i.e., constant interest rate per validator) and $p = 1$ (i.e., constant total amount of interest paid out by the protocol). In particular, given $p = 1/2$, $\gamma$ and $\beta$ were derived by reverse-engineering the constants from two desired outcomes: assuming that 10M ETH has been deposited, i) validators earn $\approx 5\%$ annual interest if everyone (nearly) always votes, and ii) if $50\%$ of the validators go offline, they lose $50\%$ of their deposits in 21 days.

**Funding:** Rewards are paid to validators by the protocol. As discussed in [43], the Casper contract was planned to receive an initial amount of 1.5M newly created ETH after a hard fork (coinciding with the changes to the fork-choice rule used by the clients). If 10M ETH were deposited, then this would keep the contract funded for roughly 2 years.\(^{15}\) This amount of ETH is intentionally kept limited to maintain an informal (i.e., dependent on further hard forks) deadline for the transition to full PoS.\(^{16}\) If the contract has insufficient funds, validators still earn interest (as their deposits are kept as contract variables), but are unable to withdraw their deposits.

**B. Overhead of the Hybrid Casper FFG Contract**

The calls to the Casper contract impact the throughput of the protocol because they use the same client bandwidth and processing power as regular transactions. In particular, we will study the contract’s consumption of gas (which, as discussed in Section II-A measures the computational load) relative to the total gas limit. The total block gas limit is variable and can be influenced by the miners, although since December 2017 it has been close to 8M gas (see Etherscan.io). The estimated gas costs of the six main types of function calls in the contract are displayed in Table III. Although the exact gas consumption of function calls depends on various external factors, including the exact numerical values of the arguments, the functionality to produce estimates is built into the Vyper compiler. Per epoch, there is ideally one `initialize_epoch` call, and one `vote` call per validator. The cost of the `deposit`, `logout` and `withdraw` functions is assumed to be negligible, in part because of the minimum time period between...
depositing and withdrawing. We assume the same for slash because of the high cost of violating a slashing condition. The load of the two other calls is unevenly distributed across the epoch: the initialize_epoch call will come early in the epoch, but most of the vote calls are expected to arrive in the later part of the epoch when the probability of voting for a ‘losing’ block in a temporary PoW chain fork is small enough. In the benchmark parametrisation, there are 50 blocks per epoch, and we assume that votes arrive in the final three quarters of the epoch, i.e., the last 37 blocks. The impact of the initialize_epoch call during the first 13 blocks is roughly equal to $742K/(13 \cdot 8M) \approx 0.7\%$, which is small. However, the impact of the votes during the last 37 blocks can be considerable. With 100 validators, the expected gas cost per epoch is roughly equal to $100 \cdot 532K/(37 \cdot 8M) \approx 18\%$. This confirms similar observations by Ethereum researchers that, even under proper protocol updates, no more than 592 (or even 400) validators could be supported.

Several approaches can be taken to limit the number of participating validators. The intended approach by Ethereum was to impose a fixed minimum deposit size of 1500 ETH. Alternative approaches would be to only accept new deposits beyond a hard limit of $N$ validators, to only accept votes from the $N$ validators with the highest deposit size, or to dynamically adjust the minimum deposit size based on the number of validators. Accurate predictions of the impact of the minimum deposit on the number of validators require economic modelling that is outside the scope of this paper. As for other PoS-based blockchain platforms: in EOS, 21 delegates chosen by the stakeholders control the consensus algorithm, whereas Cardano aims for 100 stake pools.

Off-Chain Messages: Another approach to mitigate the network load is to move hybrid Casper FFG messages onto a separate chain. As a result, two interdependent blockchains operate simultaneously: the traditional PoW chain, and a side chain called the beacon chain.

The core protocol messages (vote and slash) are then moved to the latter, and the evolution of the rewards is processed internally by clients. A contract on the main chain is still created to handle deposit, logout, and withdraw messages, and to process exchanges from ETH to/from the reward variables on the beacon chain. The initialize_epoch calls are no longer necessary as clients process the epoch transitions internally.

The advantages of this approach are the possibility of message processing optimisations (e.g., bit masks and signature aggregation), or the parallel processing of vote messages which was found to be challenging in the contract set-up, and facilitation of a transition to a sharded blockchain, by serving as the central chain connecting the various side chains. The disadvantage is that substantial modifications to the clients will be required. The block proposal/consensus mechanism on the beacon chain is still under active development — it is envisaged to use full PoS (with the same validator set as Casper FFG) in its final iteration. Given its long-term benefits, the dual-chain approach has been chosen as the way forward for Ethereum.

C. Other Issues & Limitations

We conclude with remarks of a general nature and issues that we detected from our study and the implementation of the Casper FFG contract. First, the “finder’s fee”, cf. Section II-C for detecting a violator of slashing conditions may create conflicting incentives and competition between validators. Second, in the case that the network experiences a large partition or fork, cf. Figure 4, honest validators who have voted and finalised on the non-canonical chain will sustain heavy losses to return to the main chain. Third, to focus on validators’ mechanics, we ignored potential collusions between validators and PoW miners. This point is not relevant in a pure PoS implementation and would have shifted the present analysis away from Casper’s properties of interest. Additionally, we have focussed on a static validator set in this paper and leave further analysis of dynamic validator sets to future work. Regarding new nodes needing to choose between conflicting checkpoints, we hope to investigate heuristics in future work. (In any case, this depends on the choice of bootstrapping nodes by the client, and is therefore a question of proper client implementation.) Finally, despite increasing security, the checkpointing mechanism does not reduce confirmation times (2 epochs = 100 blocks). Instead, the full benefits of Casper in terms of block-confirmation times are expected to be realised in a pure PoS implementation of the Ethereum blockchain.

V. Conclusions

In this paper, we analysed the Casper FFG contract that was evaluated on a dedicated Ethereum testnet. We described its core mechanism and showed that its incentives scheme ensures liveness whilst providing security against the finalisation of conflicting histories, i.e., checkpoints. As a finality protocol that can be overlaid on both PoW and PoS blockchains, hybrid Casper FFG can be of interest to a broad audience within the blockchain ecosystem. Our findings on liveness, safety, and implementation remain particularly relevant for Ethereum’s transition to a sharded design in which the Casper FFG philosophy is being carried over.

Acknowledgements

The authors would like to thank Pieter Hartel, Chih-Cheng Liang, Ivan Homoliak, and Vi for their helpful comments.

Table III

| Function       | Estimated gas cost |
|----------------|--------------------|
| initialize_epoch | 742 393            |
| deposit         | 831 687            |
| logout          | 131 308            |
| withdraw        | 224 155            |
| vote            | 532 031            |
| slash           | 280 864            |
REFERENCES

[1] I. Bentov, A. Gabizon, and A. Mizrahi. Cryptocurrencies without proof of work. In International Conference on Financial Cryptography and Data Security, pages 142–157. Springer, 2016.

[2] I. Bentov, R. Pass, and E. Shi. Snow White: Provably secure proofs of stake. IACR Cryptology ePrint Archive, 2016:919, 2016.

[3] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In 2015 IEEE Symposium on Security and Privacy, pages 104–121, 2015.

[4] J. Brown-Cohen, A. Narayanan, C.-A. Psysos, and S. M. Weinberg. Formal Barriers to Longest-Chain Proof-of- Stake Protocols. ArXiv e-prints, 2018.

[5] L. Brünjes, A. Kiayias, E. Koutsoupias, and A. Stouka. Reward sharing schemes for stake pools. arXiv:1807.11218, 2018.

[6] S. Buchko. What Is an EOS Delegate? A Basic Explainer, 2018. Available [online]. [Accessed: 4-12-2018].

[7] V. Buterin. Beacon chain casper FFG RJP mini-spec, 2018. Available [online]. [Accessed: 14-9-2018].

[8] V. Buterin. Casper and Sharding chain v2.1, 2018. Available [online]. [Accessed: 14-11-2018].

[9] V. Buterin. Casper FFG and light clients, 2018. Available [online]. [Accessed: 3-9-2018].

[10] V. Buterin. CBC-Casper in the face of the new PoS+Sharding plans on Ethereum [comment], 2018. Available [online]. [Accessed: 3-9-2018].

[11] V. Buterin. Ethereum 2.0 spec – Casper and sharding, 2018. Available [online]. [Accessed: 28-11-2018].

[12] J. Chen and S. Micali. Algorand.

[13] J. Choi. Casper: The fixed income approach, 2017. Available [online]. [Accessed: 3-9-2018].

[14] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Bootstrapping the Blockchain, with Applications to Consensus and Fast PKI Setup. Cryptology ePrint Archive, Report 2016/991, 2016. https://eprint.iacr.org/2016/991.

[15] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, SOSP ’17, pages 51–68, New York, NY, USA, 2017. ACM.

[16] A. Kiayias, E. Koutsoupias, M. Larangeira, L. Brünjes, D. Karakostas, and A. Stouka. Incentives and staking in Cardano, 2018. Available [online]. [Accessed: 4-12-2018].

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

[18] J. Kwon. Tendermint: Consensus without mining, 2014.

[19] W. Li, S. Andreina, J.-M. Bohli, and G. Karam. Securing Proof-of-Stake Blockchain Protocols. In J. Garcia-Alfaro, G. Navarro-Arribas, H. Hartenstein, and J. Herrera-Joancomartí, editors, Data Privacy Management, Cryptocurrencies and Blockchain Technology, pages 297–315, Cham. 2017. Springer International Publishing.

[20] C.-C. Liang. Personal communication.

[21] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[22] W. Peaster. Alpha Casper FFG testnet released, Ethereum’s PoS excitement grows, 2017. Available [online]. [Accessed: 4-12-2018].

[23] W. Peaster. One step closer: Casper testnet arrives on Parity Ethereum client, 2018. Available [online]. [Accessed: 4-12-2018].

[24] QuantumMechanic. Proof of stake instead of proof of work. Available [online]. [Accessed: 14-11-2018].

[25] J. Kwon. Tendermint: Consensus without mining, 2014. Available [online]. [Accessed: 3-9-2018].

[26] D. Ryan. Analysis of Casper PoW Reward Reduction. Available [online]. [Accessed: 4-12-2018].

[27] D. Ryan. Casper ⊙ Sharding. Available [online]. [Accessed: 3-9-2018].

[28] D. Ryan. Gas cost profiling, optimization, and management, 2018. Available [online]. [Accessed: 4-9-2018].

[29] D. Ryan. Validator Implementation Guide, 2018. Available [online]. [Accessed: 17-12-2018].

[30] D. Ryan and C.-C. Liang. Ethereum improvement proposal 1011. Available [online]. [Accessed: 3-9-2018].

[31] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In J. Grossklags and B. Preneel, editors, Financial Cryptography and Data Security, pages 515–532, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[32] T. Swanson. How much electricity is consumed by Bitcoin, Bitcoin Cash, Ethereum, Litecoin, and Monero?, 2018. Available [online]. [Accessed: 19-12-2018].

[33] T. Team Rocket. Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies, 2018. Available [online]. [Accessed: 4-12-2018].

[34] G. Wood. Ethereum: A secure decentralized generalized transaction ledger, 2014.