Authors: Joachim Neu, Srivatsan Sridhar, Lei Yang, David Tse

Preprint: https://arxiv.org/abs/2308.05096

Source code: https://github.com/tse-group/flexible-eth

TL;DR:

This post describes a new ‘super-finality

’ confirmation rule for the Casper part of Ethereum’s consensus protocol. Super-finality promises to boost the safety resilience of Casper beyond the threshold of 33\%

of stake provided by Casper’s ‘finality’ confirmation rule. Super-finality can be implemented client-side (e.g., in wallets) using information provided by the Beacon API, without requiring any changes to the protocol run by Ethereum validators. For more details, see this preprint (esp. Section 5). There is also a prototype implementation in Rust.

Overview

Recap: Confirmation Rules & Safety

Safety

in a blockchain means that once a transaction in the blockchain is considered confirmed

by the protocol, then this transaction will remain in the blockchain (as seen by everyone) forever in the same position. That is, this transaction cannot be removed, reordered, or altered. To see why safety is crucial, let’s consider an example: Alice wants to exchange 100 ETH that she holds in her ETH hardware wallet for 200,000 USD at an exchange. Alice sends a transaction to the Ethereum network, transferring 100 ETH to an address owned by the exchange, and in return, the exchange sends 200,000 USD to Alice’s bank account. Before allowing Alice to withdraw the USD, the exchange must ensure that the ETH deposit sent by Alice will remain in the blockchain forever. Otherwise, if the ETH deposit were to revert after Alice has withdrawn the USD, Alice would end up with both ETH and USD, and the exchange would end up with neither. The exchange would use a confirmation rule

to decide when a transaction is considered confirmed. The rule’s safety ensures that the exchange does not lose money. A protocol can have multiple/different confirmation rules that achieve different safety vs. latency tradeoffs. Blockchain participants can then choose a confirmation rule according to their user-experience requirements and risk appetite.

Casper’s Finality Confirmation Rule

Ethereum’s Gasper protocol is a composition of the LMD GHOST and the Casper (FFG) consensus protocols (see Figure 1 in this paper). Users who require low latency (and can tolerate the risk of reorgs) follow the tip of the LMD GHOST chain or use a confirmation rule for LMD GHOST (for instance the ones described and referenced in this other post). On the other hand, users who require safety adopt the confirmation rule of Casper, popularly known as finality

. When a block is finalized

, it is impossible to revert transactions in that block, unless Ethereum validators controlling 33\%

of the total staked ETH collude to do so

. However, recently, more than 33\%

of staked ETH are controlled by Lido, which has people worried that entities “above critical consensus thresholds pose risks to the Ethereum protocol”.

Super-finality Confirmation Rule for Casper

This post discusses a new ‘super-finality

’ confirmation rule for the Casper part of Ethereum’s consensus protocol. Super-finality allows clients to maintain safety even when more than 33\%

of stake turn malicious. The client can choose the level of safety it desires, from 33\%

all the way up to 99\%

, and can apply the rule on a per-transaction basis. For example, a client may use super-finality only on high-value transactions, and may tweak the safety level according to the transaction value. Super-finality can be implemented client-side, and does not require any fork of Ethereum or changes to the validators or full nodes. A prototype implementation in

Rust using the Beacon API is available on Github.

How is super-finality useful? In the previous example with Alice and the exchange, if the exchange sent out the USD after confirming Alice's transaction using Casper's default finality confirmation rule, then an adversary controlling 34\%

of staked ETH could remove this transaction from the blockchain and leave the exchange short of both ETH and USD. However, if the exchange waited until Alice's ETH deposit is super-finalized with a safety level of 99\%

, then even if up to 99\%

of the stake were to turn malicious, all clients would see Alice's transaction super-finalized at the 99\%

safety level.

This higher safety comes at the cost of slightly higher latency. To super-finalize a transaction with 99\%

safety, the exchange must wait for one additional epoch, in which also 99.5\%

of the validators actively vote. See Figure 1 for a comparison of the latency of Casper finality, and super-finality with different safety levels. The latency for super-finality with 99\%

safety is only 8 minutes (slightly more than one epoch) more than that of finality. Moreover, the exchange can choose to bear this delay only for very-high-value deposits.

[

A plot showing average confirmation latency for Casper finality and of super-finality with different safety levels

2410×1308 342 KB

](https://ethresear.ch/uploads/default/original/2X/c/ccfb3d090155de4c5a90535fc25bb93e7e5f1db4.png)

Users opting for very high safety level (e.g., 98\%

) may see brief stalls of the chain, when participation of validators dips too low (this can be aggravated, for instance, during a hard fork—hence the upward tick at high safety levels in Figure 1). However, usually, the participation by Ethereum validators is high, so that super-finality promptly catches up with finality and even very high safety does not incur significantly higher confirmation latency, see Figure 2.

[

A plot showing the Ethereum mainnet chain confirmed by Casper finality (33% safety level) and by super-finality with 33% and 98% safety levels

1920×1213 198 KB

](https://ethresear.ch/uploads/default/original/2X/b/b870d483edf4dd8f06f2b3edf0c4b8171c6f73de.jpeg)

Construction (High Level)

For further technical details, see this preprint (esp. Section 5).

Super-finality uses votes (a.k.a. 'attestations

' in Ethereum) cast by validators as part of Ethereum'sCasper component, and confirms blocks as follows:

Super-finality rule (see Figure 3):

Super-finalize block A

(and its ancestors) iff: (1) we see a block B

whose state commits to A

(or a descendant of A

) as the finalized checkpoint

, and (2) we see votes in favor of B

(i.e., FFG votes with target B

or a descendant of B

) from validators totalling q

fraction of stake (for instance, q=90\%

).

Here, for (1), recall that the finalized checkpoint is a part of the state commitment of the beacon chain's state transition function, and note that necessarily B

will be a descendant of A

. For (2), these votes could be either received on the peer-to-peer network, or included as payload of blocks in the protocol's block-tree.

[

Illustration of the super-finality rule

2606×608 49.8 KB

](https://ethresear.ch/uploads/default/original/2X/1/193c29eb99ccd7d95787c54eb6f4386cb0332e19.png)

Key Idea #1:

Higher Confirmation Quorum for Higher Safety Resilience

Super-finality is based on two key ideas. First, we keep the quorum size for finality (which is used by validators to decide for what blocks to vote) fixed at 2/3

, and allow clients to individually choose a higher quorum size to confirm transactions with super-finality. Increasing the quorum size leads to higher safety due to quorum intersection

. To see why, suppose that there are two different blocks A

and C

both of which get votes from 2/3

fraction of the stake. Then, considering the total stake as 1

, it is clear that at least 2/3 + 2/3 - 1 = 1/3

fraction of the stake voted for both blocks (see Figure 4 below). That explains finality's safety threshold. When we increase the quorum q

beyond 2/3

(e.g., q=90\%

), the fraction of stake that must vote for both blocks to confirm both of them increases (e.g., 2q-1 = 80\%

), giving a glimpse of why super-finality achieves higher safety.

[

Illustration of quorum intersection

2592×786 219 KB

](https://ethresear.ch/uploads/default/original/2X/f/f940636c147d5c0043f9d2ac6a491e53d7900636.png)

Key Idea #2:

A Validator's Vote Reveals what Finalization it is Locked on

The second idea is an observation that once an Ethereum validator sees a block as finalized (as per 2/3

votes), it will never vote for any blocks that are inconsistent with the finalized block. Therefore, if a validator votes for a block B

which commits to block A

as the finalized checkpoint (i.e., as part of the state commitment of the beacon chain's state transition function), then this

validator signals that it has verified the finalization of A

. This validator shall thus never vote for a block C

(or its descendants) that conflicts with A

(see Figure 5). Thus, when a client sees validators with q

fraction of stake vote for block B

, thereby super-finalizing A

, it knows that super-finalizing C

(with same or higher quorum) would require validators with at least 2q-1

fraction of stake to be malicious (since they must vote for conflicting blocks, which an honest validator would never do).

[

Illustration of Ethereum validators locking on to their finalized checkpoint

2570×962 205 KB

](https://ethresear.ch/uploads/default/original/2X/9/9c2395bf165c57aeeb42f7c4480c886d5bd68cfb.png)

The above two ideas ensure that for two clients to super-finalize conflicting blocks with q=90\%

quorums each, at least 2q - 1 = 80\%

stake must have been malicious. Thus, 80\%

is their safety level. Additionally, for a third client to super-finalize a conflicting block with a 67\%

quorum, at least 90\% + 67\% - 100\% = 57\%

stake must have been malicious, which is more than the 33\%

guaranteed by Casper's default finality. The cost of super-finalizing blocks with a high quorum (e.g., q=90\%

) is that if validators with 1-q = 10\%

of the total stake go offline, then no block receives q = 90\%

votes, so no block gets super-finalized, i.e., the client loses liveness

. This safety/liveness trade-off has been proven to be fundamental, and super-finality achieves the optimal trade-off between the safety level (e.g., 2q-1 = 80\%

) and liveness level (e.g., 1-q = 10\%

).

Prototype Implementation

Ethereum consensus-client software exposes all the data required to implement super-finality, namely finalized checkpoints and attestations, through the Beacon API. We implemented a prototype of super-finality as a standalone program that subscribes to an unmodified consensus client for chain data, and outputs the tip of the chain super-finalized at a desired safety level. The source code is available on Github. The program uses the /states/finality_checkpoints

endpoint (of the Beacon API) to query finalized checkpoints, the /states/committees

endpoint to query the set of validators selected to vote in a given epoch, and the /blocks

endpoint to fetch blocks and examine the attestations included inside. We tested the prototype with Prysm and Lighthouse.

Applications such as wallets, custodians, or exchanges can easily adopt super-finality. Ethereum execution clients allow applications to specify a block hash when querying blockchain state through the JSON-RPC, and will answer the query based on the blockchain state as of the block with that block hash. To use super-finality, we simply specify the hash of the block confirmed by super-finality when making queries, so the execution client will answer the queries as per the chain confirmed by super-finality. To make this process transparent to applications, one can use a JSON-RPC proxy such as Patronum, which will listen to the implementation of the super-finality rule for the latest super-finalized block hash, and automatically rewrite application JSON-RPC queries to use this block hash (see Figure 8 in the preprint).