Constraining Fork-Choice with Ethereum's Casper FFG: Understanding and surveying an irrevocable consensus mechanism for blockchains

Guillaume Paya-Monet
Bachelor École polytechnique

September - December 2019

Contents

Abstract

In this research project, in the context of CSE303 Computer Science project, we will look at Ethereum's new consensus Casper the Friendly Finality Gadget (FFG). Casper has been introduced as a transition into Ethereum 2.0 (otherwise known as Serenity) and will enable a proof-of-stake (PoS) based finalisation in the Ethereum blockchain. Furthermore, in this paper, we will take a look at how Ethereum functions with its proof-of-work (PoW) and introduce the new system as a mean of finalisation and limit forking in the chain by forcing irrevocability in a classical consensus, since PoW does not achieve irrevocability.

Keywords: *Blockchain, Consensus, Proof-of-Stake (PoS), Proof-of-Work (PoW), Ethereum, Casper FFG*

2 Context

Before entering deep in the blockchain technology, it is first important to introduce its context. First of all, what is blockchain? In short, it is all around us, whether it be in digital currencies, in securities or for record keeping, namely ownership and voting. Blockchain is most commonly referred to as a "decentralized, distributed public ledger" but what does it actually mean? A distributed ledger means that a database is shared across multiple participants, in this case, most chains are accessible by any public participant who wish to explore any transaction that ever happened and contribute in a specific blockchain. The main advantage of distributed public ledgers is that an intermediary, like a bank, is no longer needed for logging purchases. The challenge comes in having a secure consensus since there is no trusted third-party.

As the name entails, in this record-keeping technology, information (such as transactions) is stored in form of a "block". The metadata of a block are as followed:

- Version: A version number for software upgrades.
- Merkle root: A hash of a hash of all the transactions.
- Hash of the previous block: A reference to the hash of its parent block in the chain.
- Timestamp: the time of creation of the block.
- Nonce: A counter for the PoW algorithm. *(Only for PoW consensus)*
- Difficulty: A target difficulty to regulate the PoW algorithm. *(Only for PoW consensus)*

Furthermore, depending on the blockchain there exists different kinds of consensus to add those blocks. We will look at Proof-of-Work (PoW) and Proof-of-Stake (PoS) but there are others such as the Delegate Proof-of-Stake (DPos). Of course they all come with their respective pros and cons and need to be adapted to their respective technologies.

3 Aims

Ethereum is a major blockchain and it is currently about to undergo a massive change in its algorithmic consensus, switching to a PoS approach. It is crucial for any player to comprehend what those changes represent.

This research will allow Pr Daniel AUGOT and the researchers in the GRACE Laboratory at École polytechnique, dedicated to cryptography, to understand the main components of Casper FFG. By doing so, they will be able to diagnose quickly if a deeper analysis is required in some parts of their research.

Also, this paper presents in an accessible way one of the forefronts of the current state of the art in consensus technologies for distributed ledgers, so that it can support the cryptography community in their efforts to advance knowledge in the field.

At a more personal level, I chose this research project as, having worked on Ethereum-related projects this summer as part of the Alyra Blockchain Development Summer School, I wanted to continue specifically focusing on Ethereum. The Alyra projects had required me to create Smart Contracts for various situations, I wanted then to deepen my understanding of how everything fits together.

4 Research

The second biggest blockchain and crypto-currency at the moment is known as Ethereum and respectivelty Ether. It was released in July 2015 by authors Vitalik Buterin, Gavin Wood and Joseph Lubin. What is special about this blockchain is that it has its own object-oriented language called *solidity* which enables individuals to create programs which are themselves stored on the blockchain and automatically executed by miners. Ethereum will soon move towards Ethereum 2.0, otherwise known as Serenity, as means to increase its security and scalability by using a more PoS based consensus.

The first step into this change is with their new consensus Casper the Friendly Finality Gadget (Casper FFG).

4.1 Ethereum Casper FFG

This research project will look at Casper FFG and use it to identify the reason why Ethereum wants to move towards Proof-of-Stake compared to its current consensus of Proof-of-Work.

Though to avoid confusion, Casper FFG does not remove completely PoW but rather adds a layer of PoS on it.

4.2 Research context

First it is important to lay out how Proof-of-Work and Proof-of-Stake work considering they are the prime actors in this system. As stated above, participants will add blocks of information onto the blockchain, and the way they achieve it is via those two consensus mechanisms.

In Ethereum Proof-of-Work, participants, otherwise known as miners, will win the right to add blocks to the blockchain according to their computational power. The difficulty measure is adjusted every so often. Miners will use the Ethereum hash function Ethash, to hash all the information stated in **2) Context**. Upon reaching a certain hash, then it will be compared to the difficulty, if the calculated hash is smaller than the difficulty, then the block can be proposed to the network. On the other hand, if the hash is bigger than the difficulty, then the nonce has to be incremented, and the hash is calculated again. The process is then repeated again until the hash is smaller than the difficulty. The miner who manages to solve this "puzzle" is then rewarded a static block reward as an incentive to contribute to the blockchain.

In Proof-of-Stake competition is removed as the creator of the block is chosen algorithmically based on the amount of money a participant has deposited at some point. Then the participant will receive a reward and gain the transaction fee that has occurred, the *gas price*. Gas is used to measure the computational power that will be required to mine a certain transaction. A system that is difficult to influence as it would require participants to own more than 50% of the total amount of currencies.

4.3 Situation Analysis

Unfortunately, with this current proof-of-work based Ethereum system in place, there are many flaws coming with it:

- **Irrevocability / Potential hacking**: Considering the probability of mining success is proportional to computation power, then if an entity holds 51% of the total computational capacities, he can revert transactions by creating forks. Then the systems becomes revocable.
- **Loss of information**: With the heavy competitions in mining blocks, it could happen that two blocks are produced at the same time. But Information stored in one is not the same as the other.The network will eventually resolve the split by accepting the chain that has accumulated the highest proof-of-work, meaning the highest nonce. Resulting in the information stored in the rejected block to disappear, thus removing evidence of any transactions or other actions.
- **Power usage**: Since PoW functions entirely based on computational power, it results in large datacenters being used to increase the mining power. Though mining is considered free, a large amount of energy is then consumed to run the process.

4.4 Research Hypothesis

The hypothesis that will be tested in this project is that the specific protocol Casper the Friendly Finality Gadget will help for a smooth transition into Ethereum 2.0. A smoother transition by implementing a more PoS based consensus. Adding upon that hypothesis, Casper FFG will also help prevent and stabilise fork choice situations arising from PoW and ensure an additional safety on the Ethereum blockchain.

4.5 Methodology

To test this hypothesis, two steps will be done:

1. Read the papers written by the creators of Ethereum and decrypt how Casper FFG works and what it means for the blockchain. This is giving us the major part of the analysis and it significance for the future of blockchain in the direction of travel for one of its major player.
2. Look at an implementation of Casper CBC to extract the state machine that describes the protocol. The theoretical approach is completed by a surface analysis of an implementation of Casper CBC. This provides a practical view of the situation that complements and clarify how particular theoretical points can be implemented.

5 Findings

5.1 Casper FFG's mechanism:

What is unique in Casper the Friendly Finality Gadget is that the PoS is introduced with the use of checkpoints. Nodes in the network assume the role of validators by depositing tokens on the PoW chain. Once, then, has to wait a period of minimum 120 days before being able to stop being a validator. Validators will then vote for specific blocks, checkpoints, where the number of said blocks is $i \cdot l$, where i is the *epoch number* of the checkpoint and l is the *epoch length* ($l = 50$ in Casper FFG).

To cast a valid vote Validators will require the following data to be sent to the network:

- v the validator index.

- t hash of the target checkpoint.
- $h(t)$ height of the target checkpoint in the checkpoint tree.
- $h(s)$ height of a justified source checkpoint, s needing to be an ancestor of t.
- S the signature of $< s, t, h(t), h(s) >$ using the validator's private key.

The goal of validators is to *justify* and *finalise* the checkpoints they believe are the most suited in the event of forking in the blockchain. There are then two stages in the process for finalisation:

1. **Justification**: Checkpoints are justified by the validators if 2/3 of the vote, in terms of stakes deposited by the validators, were for that checkpoint at a specific epoch. Using the voting input (seen at the beginning of 5.1), here s has so be a justified ancestor of t, for t to be potentially justified in its turn.
2. **Finalisation**: Checkpoint is then finalised after justification if its direct child is justified in turn. Then that checkpoint becomes automatically finalised. The genesis checkpoint is by rule both justified and finalised. Conflict can arise if there are two finalised checkpoint such that neither is an ancestor of the other.

As a result of voting, there will then be three different outcomes.

- **Correct voting** Assuming the vote is correct in its input, then when a checkpoint is being finalised at a certain epoch, the corresponding 2/3 validator's stake that have voted for the appropriate block will have their deposit increase by a positive interest rate. The interest depends on the total deposit brought by the validators. If checkpoints are not finalised, then the deposit remains the same.
- **Non voting** Validators will be penalised and have their deposit shrunk if they do not vote during an epoch. The size of penalty is proportional to the number of validators who do not vote.
- **Conflicting/Incorrect voting** There are severe consequences for validators who are caught voting conflictingly. Their deposit could then be partially or completely removed. Removing deposit is called *slashing*. The votes of validators who vote incorrectly are ignored and the caster will be considered as a non-voter.

5.2 Experimentation

To study the paper, I decomposed the paper by parts and studied the individual sections bit by bit, to be sure of each section. The list of section are as followed:

1. Understand PoW and PoS
2. Identify the current flaws of Ethereum with PoW
3. Understand the formalism with regards to representing connection between blocks.
4. Validators and voting
5. Justified and finalisation

Unfortunately, there is no implementation of Casper FFG available on GitHub which is why the code for Casper CBC was used. The Casper CBC code [3] that was analysed has not been produced by the Ethereum team but is based on the "correct-by-construction" consensus protocol abstract produced by the Ethereum Foundation.

CBC works in a way in which the protocol evolves dynamically to fit the properties, define at the beginning, that the protocol must specify. Looking at the implementation of Casper CBC was appropriate considering the differences between CBC and FFG are immaterial to the point being made. In the end, Ethereum 2.0 will be influenced by both Casper FFG and Casper CBC.

The exhaustive approach to draw a class map of the implementation would be to run by a dynamic system analysis. As such analysis was difficult with open source tools and always ended in a segmentation fault without giving result due to interference of the analysis with the implementation, a static analysis was completed with point addition of direct look at the code itself to understand some apparent breaks in the map.

5.3 Results

5.3.1 Result from paper

From the paper, we can infer that this new system only works as intended when there is at least 2/3 of honest participants in terms of stake. Honest participants would infer that they vote for the benefit of the blockchain and do not try to create competing chains. Otherwise, either the network will not advance as no checkpoints will be justified, or competing chains could be create and the network will be forced to converge to a malicious path.

An important theorem to remember is **Theorem 5** [1]. The theorem states that in a non forking condition, then conflicting checkpoints are finalised when at least 1/3 of the validators violate the slashing conditions. The slashing conditions are called the "minimal slashing conditions" and they are there to prevent the validators from influencing the blockchain with negative repercussions.

5.3.2 Result from graphs

When the provided Casper CBC is ran, a graph shown in *Figure 1* is outputted, it shows connection between validators. But what is interesting to see is the static analysis of the code as shown from *Figure 2, 3 and 4*.

The first graph of call functions (shown in *Figure 2*) is the representation of what is needed to be called to plot the graph produced in *Figure 1*. For that, three functions are called:

- **Abstract view**: this system manages the input/output when given a specific protocol to work with.
- **Message validator**: this in turn is organised in three pieces, the validator set, the validator and the messages. Every validator will in turn hash access a message generated and hash it.
- **Utils**: a tool to plot graphs.

The second graph (as shown in *Figure 3*) represents the implementation of the protocol:

- **Protocol**: In the clustered elements of the protocol, we see the structuration of BlockChain class, and the Block that links a number of classes to implement views on types and shards.
- **Sharding and Forking**: The sharding protocol, sharding block, sharding view, sharding fork choice are in a cluster of classes that are closely related to the Casper protocol. This is the key information expected from the theory of the Casper protocol itself.

The functions shown in *Figure 4*, the oracle, is to check some boundaries on estimates and is only used for implementation but not in the theory.

6 Conclusion

In this report, we have seen the reason why Ethereum wants to implement the new Casper FFG protocol. If a validator were to violate minimal slashing conditions, then their deposit would then be slashed. This ensures that the consensus works well as negative behaviour has repercussions. Making the consensus a trusted permissionless one, as anybody can join the network, but participants are forced to contribute to its upkeep. Casper FFG can then improve the finalisation of the blockchain as validators are forced to stand by their choices. Casper FFG is, therefore, an optimal solution for Ethereum. However, with this system, instead of better opportunities, the rich get richer.

References

[1] Vitalik Buterin, Daniel Reijsbergen, Stefanos Leonardos, Georgios Piliouras. *Incentives in Ethereum's Hybrid Casper Protocol* - 11 March 2019
`https://arxiv.org/pdf/1903.04205.pdf`

[2] Vitalik Buterin and Virgil Griffith. *Casper the Friendly Finality Gadget*
`https://arxiv.org/pdf/1710.09437.pdf`

[3] Ethereum's code for CBC casper
`https://github.com/ethereum/cbc-casper`

8 Figures

Figure 1: Output of the code with 10 validators

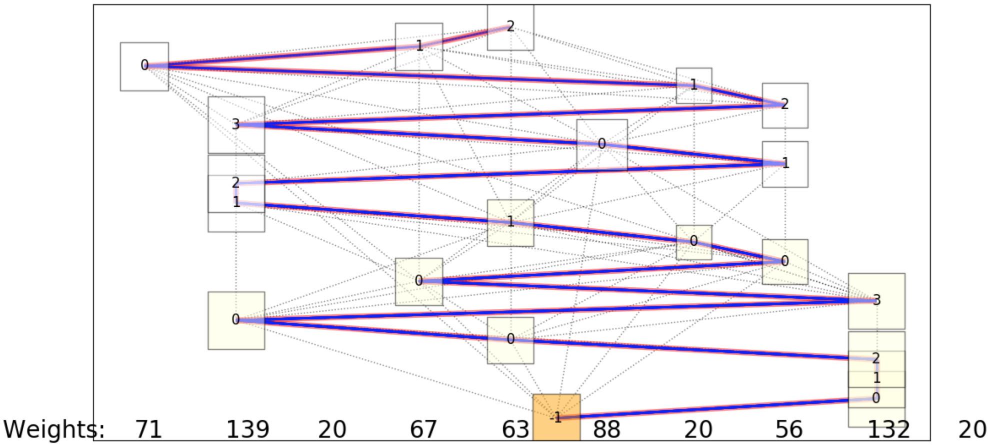


Figure 2: Calls for plotting the output graphs produced

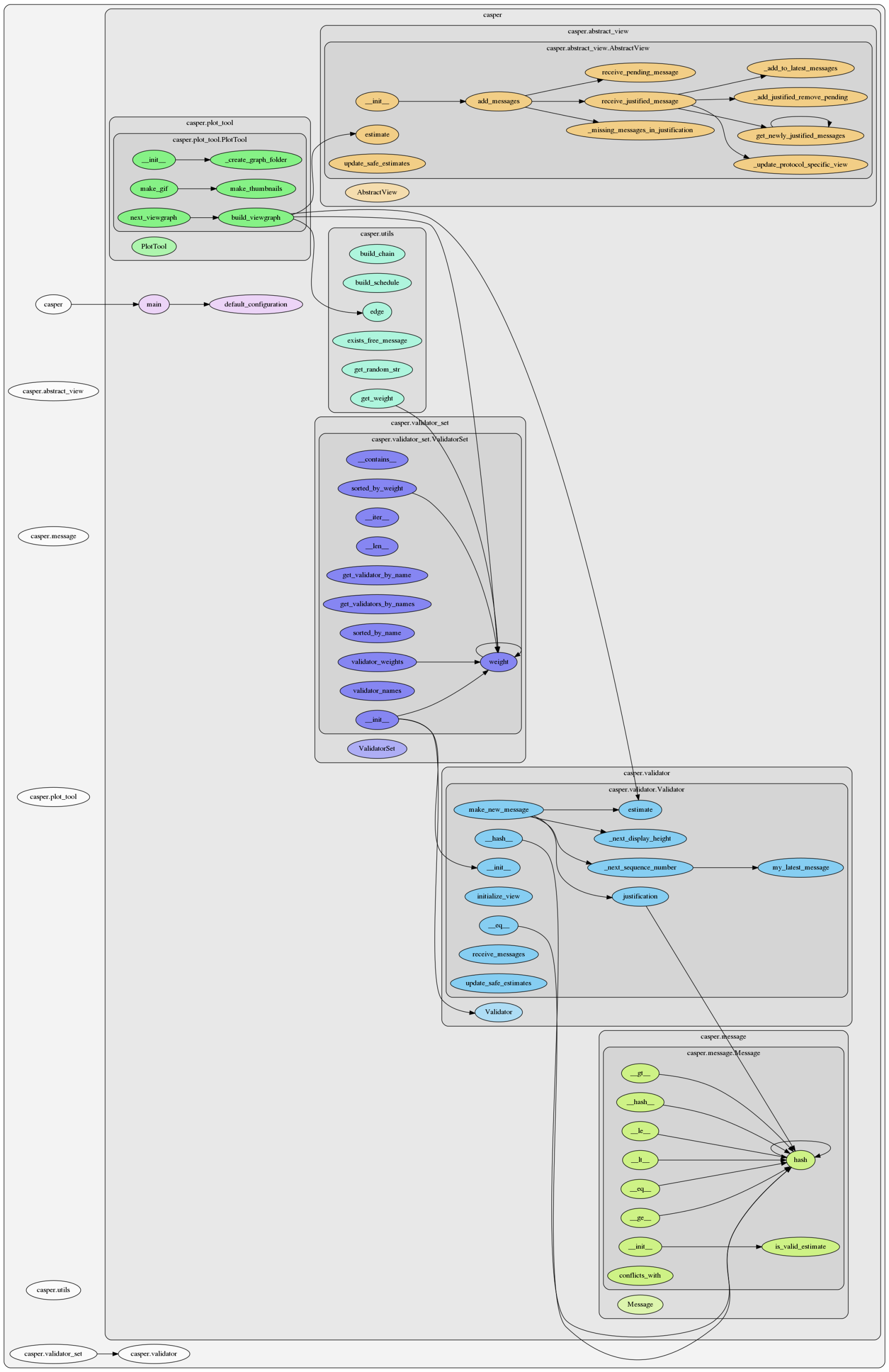


Figure 3: Calls for the implementation of the protocol

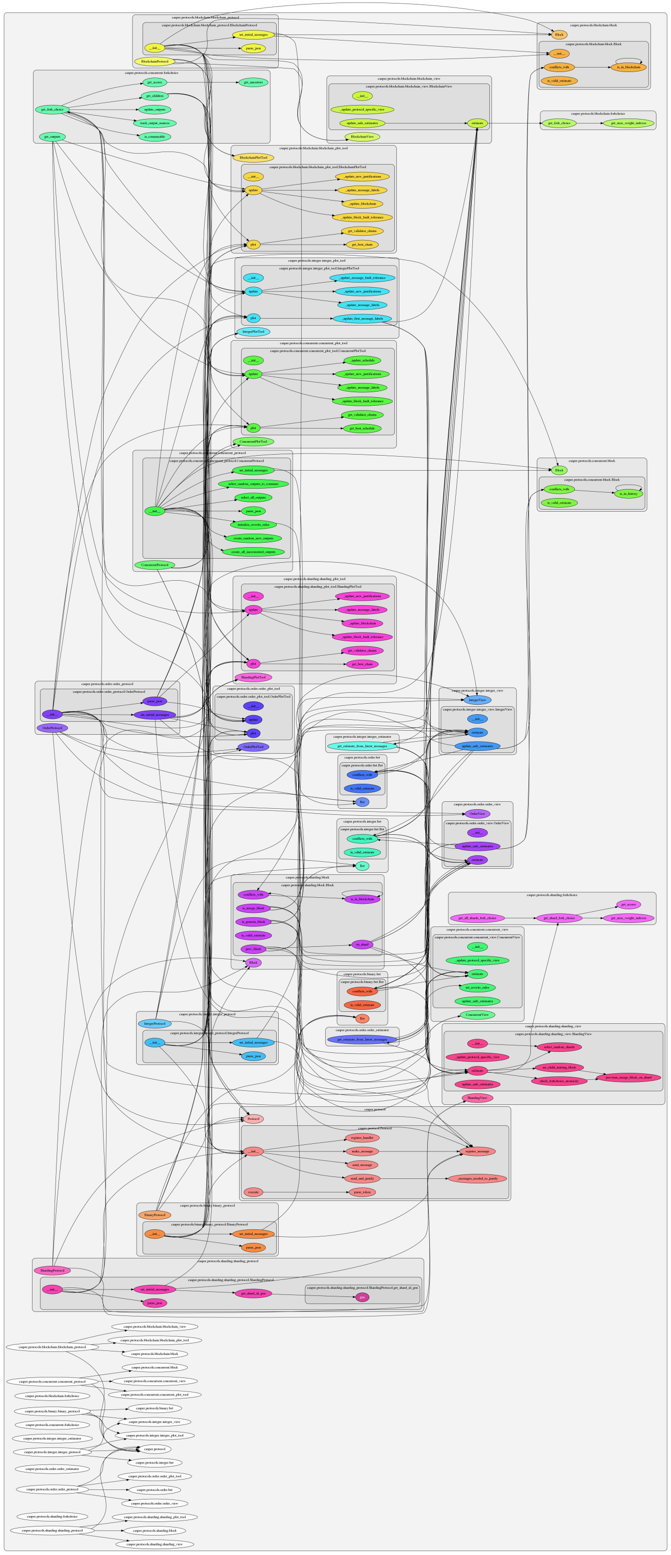
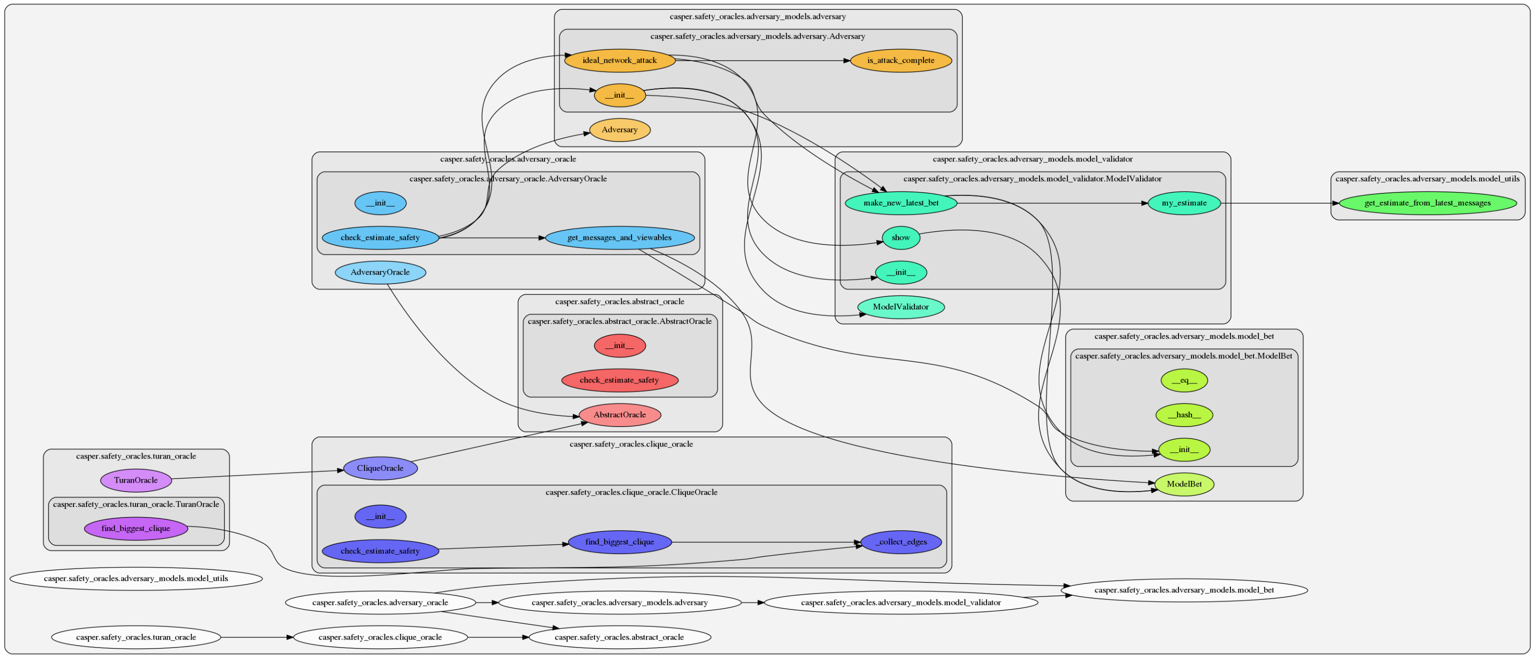

casper.protocols.blockchain.blockchain_protocol
casper.protocols.blockchain.blockchain_protocol.BlockchainProtocol
__init__
set_initial_messages
parse_json
BlockchainProtocol
casper.protocols.blockchain.block
Block
casper.protocols.blockchain.block.Block
__init__
conflicts_with
is_in_blockchain
is_valid_estimate
casper.protocols.concurrent.forkchoice
get_scores
get_ancestors
get_children
get_fork_choice
update_outputs
track_output_sources
is_consumable
get_outputs
casper.protocols.blockchain.blockchain_view
casper.protocols.blockchain.blockchain_view.BlockchainView
__init__
_update_protocol_specific_view
update_safe_estimates
estimate
BlockchainView
casper.protocols.blockchain.forkchoice
get_fork_choice
get_max_weight_indexes
casper.protocols.blockchain.blockchain_plot_tool
BlockchainPlotTool
casper.protocols.blockchain.blockchain_plot_tool.BlockchainPlotTool
__init__
update
plot
_update_new_justifications
_update_message_labels
_update_blockchain
_update_block_fault_tolerance
get_validator_chains
get_best_chain
casper.protocols.integer.integer_plot_tool
casper.protocols.integer.integer_plot_tool.IntegerPlotTool
__init__
update
plot
_update_message_fault_tolerance
_update_new_justifications
_update_message_labels
_update_first_message_labels
IntegerPlotTool
casper.protocols.concurrent.concurrent_plot_tool
casper.protocols.concurrent.concurrent_plot_tool.ConcurrentPlotTool
__init__
update
plot
_update_schedule
_update_new_justifications
_update_message_labels
_update_block_fault_tolerance
get_validator_chains
get_best_schedule
ConcurrentPlotTool
casper.protocols.concurrent.concurrent_protocol
casper.protocols.concurrent.concurrent_protocol.ConcurrentProtocol
__init__
set_initial_messages
select_random_outputs_to_consume
select_all_outputs
parse_json
initialize_rewrite_rules
create_random_new_outputs
create_all_incremented_outputs
ConcurrentProtocol
casper.protocols.concurrent.block
Block
casper.protocols.concurrent.block.Block
conflicts_with
is_in_history
is_valid_estimate
casper.protocols.sharding.sharding_plot_tool
casper.protocols.sharding.sharding_plot_tool.ShardingPlotTool
__init__
update
plot
_update_new_justifications
_update_message_labels
_update_blockchain
_update_block_fault_tolerance
get_validator_chains
get_best_chain
ShardingPlotTool
casper.protocols.order.order_protocol
casper.protocols.order.order_protocol.OrderProtocol
__init__
parse_json
set_initial_messages
OrderProtocol
casper.protocols.order.order_plot_tool
casper.protocols.order.order_plot_tool.OrderPlotTool
__init__
update
plot
OrderPlotTool
casper.protocols.integer.integer_view
IntegerView
casper.protocols.integer.integer_view.IntegerView
__init__
estimate
update_safe_estimates
casper.protocols.integer.integer_estimator
get_estimate_from_latest_messages
casper.protocols.order.bet
casper.protocols.order.bet.Bet
conflicts_with
is_valid_estimate
Bet
casper.protocols.order.order_view
OrderView
casper.protocols.order.order_view.OrderView
__init__
update_safe_estimates
estimate
casper.protocols.integer.bet
casper.protocols.integer.bet.Bet
conflicts_with
is_valid_estimate
Bet
casper.protocols.sharding.block
casper.protocols.sharding.block.Block
conflicts_with
is_in_blockchain
is_merge_block
is_genesis_block
is_valid_estimate
prev_block
on_shard
Block
casper.protocols.sharding.forkchoice
get_all_shards_fork_choice
get_shard_fork_choice
get_scores
get_max_weight_indexes
casper.protocols.concurrent.concurrent_view
casper.protocols.concurrent.concurrent_view.ConcurrentView
__init__
_update_protocol_specific_view
estimate
set_rewrite_rules
update_safe_estimates
ConcurrentView
casper.protocols.binary.bet
casper.protocols.binary.bet.Bet
conflicts_with
is_valid_estimate
Bet
casper.protocols.order.order_estimator
get_estimate_from_latest_messages
casper.protocols.integer.integer_protocol
IntegerProtocol
casper.protocols.integer.integer_protocol.IntegerProtocol
__init__
set_initial_messages
parse_json
casper.protocols.sharding.sharding_view
casper.protocols.sharding.sharding_view.ShardingView
__init__
select_random_shards
_update_protocol_specific_view
estimate
set_child_starting_block
check_forkchoice_atomicity
previous_merge_block_on_shard
update_safe_estimates
ShardingView
casper.protocol
Protocol
casper.protocol.Protocol
__init__
register_handler
make_message
send_message
send_and_justify
register_message
_messages_needed_to_justify
execute
parse_token
casper.protocols.binary.binary_protocol
BinaryProtocol
casper.protocols.binary.binary_protocol.BinaryProtocol
__init__
set_initial_messages
parse_json
casper.protocols.sharding.sharding_protocol
ShardingProtocol
casper.protocols.sharding.sharding_protocol.ShardingProtocol
__init__
set_initial_messages
get_shard_id_gen
parse_json
casper.protocols.sharding.sharding_protocol.ShardingProtocol.get_shard_id_gen
gen
casper.protocols.blockchain.blockchain_protocol
casper.protocols.blockchain.blockchain_view
casper.protocols.blockchain.blockchain_plot_tool
casper.protocols.blockchain.block
casper.protocols.concurrent.concurrent_protocol
casper.protocols.concurrent.block
casper.protocols.concurrent.concurrent_view
casper.protocols.concurrent.concurrent_plot_tool
casper.protocols.blockchain.forkchoice
casper.protocols.binary.binary_protocol
casper.protocols.binary.bet
casper.protocols.integer.integer_view
casper.protocols.integer.integer_plot_tool
casper.protocols.concurrent.forkchoice
casper.protocols.integer.integer_estimator
casper.protocols.integer.integer_protocol
casper.protocol
casper.protocols.integer.bet
casper.protocols.order.order_estimator
casper.protocols.order.order_protocol
casper.protocols.order.order_plot_tool
casper.protocols.order.bet
casper.protocols.order.order_view
casper.protocols.sharding.forkchoice
casper.protocols.sharding.sharding_protocol
casper.protocols.sharding.sharding_plot_tool
casper.protocols.sharding.block
casper.protocols.sharding.sharding_view

Figure 4: Oracle


casper.safety_oracles.adversary_models.adversary
casper.safety_oracles.adversary_models.adversary.Adversary
ideal_network_attack
is_attack_complete
__init__
Adversary
casper.safety_oracles.adversary_oracle
casper.safety_oracles.adversary_oracle.AdversaryOracle
__init__
check_estimate_safety
get_messages_and_viewables
AdversaryOracle
casper.safety_oracles.adversary_models.model_validator
casper.safety_oracles.adversary_models.model_validator.ModelValidator
make_new_latest_bet
my_estimate
show
__init__
ModelValidator
casper.safety_oracles.adversary_models.model_utils
get_estimate_from_latest_messages
casper.safety_oracles.abstract_oracle
casper.safety_oracles.abstract_oracle.AbstractOracle
__init__
check_estimate_safety
AbstractOracle
casper.safety_oracles.adversary_models.model_bet
casper.safety_oracles.adversary_models.model_bet.ModelBet
__eq__
__hash__
__init__
ModelBet
casper.safety_oracles.turan_oracle
TuranOracle
casper.safety_oracles.turan_oracle.TuranOracle
find_biggest_clique
casper.safety_oracles.clique_oracle
CliqueOracle
casper.safety_oracles.clique_oracle.CliqueOracle
__init__
check_estimate_safety
find_biggest_clique
_collect_edges
casper.safety_oracles.adversary_models.model_utils
casper.safety_oracles.adversary_oracle
casper.safety_oracles.adversary_models.adversary
casper.safety_oracles.adversary_models.model_validator
casper.safety_oracles.adversary_models.model_bet
casper.safety_oracles.turan_oracle
casper.safety_oracles.clique_oracle
casper.safety_oracles.abstract_oracle