This is a proposal for an eth1 engine based on geth, called Catalyst. Thanks to @mkalinin and @djrtwo for their valuable input.

This post presents a general overview of Catalyst, an eth1 engine[1] based on geth.

Catalyst is the name of a fictional device in the Mass Effect series, that triggered the creation of a synthetic foe called the Reapers, who in turn created the Geths.

Architecture

[

figure1

1315×453 16.5 KB

](https://ethresear.ch/uploads/default/original/2X/0/028ef58b1081be5748afa27ecc3824d63d1c2251.png)

Figure 1

: eth1-aware nodes are made of a eth2-eth1 pair in which the eth2 client manages block propagation and the eth1 “engine” manages the state transition.

Each eth1-capable eth2 client has a copy of Catalyst running by its side. They communicate through an RPC API[2]. During its lifecycle, a block goes through four steps:

1 - block production

When the eth2 client determine that it has been elected to produce the next block:

- it calls produceBlock(head)

on its Catalyst instance;

- Catalyst produces a new block on top of the specified head block, and returns it to the eth2 client;
- the eth2 client publishes the new block on the eth2 network, to be validated and added to the chain.

Note that if head

is not the eth1 chain’s current head, it will return an error.

2 - block validation

When the eth2 client has been selected to validate a block:

- it recovers this block from the network and extracts the eth1 block from it;
- it calls validateBlock

on the eth1 engine with this eth1 block as a parameter;

- Catalyst executes this block on top of its current head block and returns true

upon success and false

otherwise;

- the eth2 client votes for the block validation based on this return value.

Note that since validateBlock

is fairly redundant with insertBlock

, it might be deprecated after further experimentation with this API.

3 - block insertion

When any eth2 block is published, the eth2 client calls insertBlock

on its eth1 engine to have this block added to the chain. The block is processed and is saved to the DB along with its post state.

If the block's parent is not known, an error will be returned.

4 - setting the head

Every time the eth2 fork choice rule triggers a new head, the eth2 client calls setHead

to tell the eth1 engine that this block is the new head of the chain. This might cause a reorg.

Figure 2

in the case where the new head H^2 is on a different branch than the current head H^1, the heads is rewound from H^1 to the fork point F

, and then advanced back to the new head. The new head location is then saved.

When setting the new head H^2, the eth1 engine will trace back the parent chain, expecting to encounter either:

- the current head H^1; or
- the point at which the chain the new head is on forks from the branch that the current head H^1 is on (hereby called F

).

Figure 3

in the case where the new head H^2 is a direct parent of the current head H^1, F

is set to be H^1, and then the transition function Υ is executed (if necessary) on each block between H^1 to H^2. Then the new head location is saved.

References

1. Danny Ryan's post on Eth1+eth2 client relationship
2. Mikhail Kalinin's RPC spec document