Using total difficulty threshold for hardfork anchor: what could go wrong?

[by TXRX Team, @zilm and @mkalinin, evaluation code is here]

In the upcoming Merge Fork which encloses the current PoW chain into the Beacon chain as an Execution Layer, total difficulty was chosen as a point of transition instead of a block number (see Terminal total difficulty vs block number). This approach negotiates the risk of forking a wrong chain backed by minor hash-power but could lead to other risks during transition. In this post we are going to measure these risks and propose changes to fork schema if any is required to decrease the possibility of unforeseen transition flow.

Happy flow

It's expected that merge client settings will be known in a month before merge happens, so client developers could update their software and validator owners could update their nodes. At some announced epoch number, say, MERGE_FORK_EPOCH

the action begins: Beacon chain nodes calculates votes in the last voting period to get some major-backed Eth1Data

with block_hash

reference to the block in the Mainnet. The corresponding block is called anchor block and becomes a point from which merge total_difficulty

on PoW chain side is calculated with the following formula (seespec):

def compute_transition_total_difficulty(anchor_pow_block: PowBlock) -> uint256: seconds_per_voting_period = EPOCHS_PER_ETH1_VOTING_PERIOD * SLOTS_PER_EPOCH * SECONDS_PER_SLOT pow_blocks_per_voting_period = seconds_per_voting_period // SECONDS_PER_ETH1_BLOCK pow_blocks_to_merge = TARGET_SECONDS_TO_MERGE // SECONDS_PER_ETH1_BLOCK pow_blocks_after_anchor_block = ETH1_FOLLOW_DISTANCE + pow_blocks_per_voting_period + pow_blocks_to_merge anchor_difficulty = max(MIN_ANCHOR_POW_BLOCK_DIFFICULTY, anchor_pow_block.difficulty)

return anchor_pow_block.total_difficulty + anchor_difficulty * pow_blocks_after_anchor_block

At that moment we expect that ETH1_FOLLOW_DISTANCE + pow_blocks_per_voting_period

is already passed, so we need to skip only pow_blocks_to_merge

until the merge happens and we use anchor block difficulty as some expected difficulty at the whole set of Mainnet blocks, both passed after anchor and those coming in the next 7 days (expected value of the TARGET_SECONDS_TO_MERGE

constant). Also we need to turn on terminal block expectation logic on the Beacon chain side to be ready for the merge in advance.

Obviously, every block on this way will have different difficulty compared to the anchor and the merge will not happen exactly in 7 days. Moreover, block difficulty fluctuation is not the only stranger.

It's a vote time

Our first close attention is anchor detection on the Beacon chain side. Though slot time is perfectly sized here, we have vote intervals set by EPOCHS_PER_ETH1_VOTING_PERIOD

. We are guaranteed that MERGE_START_EPOCH

is at the start of one. Here we could get 2 possible bad behaviors:

- there is no major voted block
- votes are stalled at some obsolete block from the past

Let's check the whole Beacon chain statistics to see how often each of them happens.

Since the beginning of Beacon chain there was no such period, the least votes share was 0.66845703125, or 66.85%. Now let's check for how long votes could stall historically:

There were about 50 periods when votes were stalled but never one followed another, so the next vote period after stalled brings in the correct PoW chain reference.

Could this change during the transition period? We don't expect malice behavior from Beacon chain nodes as it negates the Proof of Stake idea and the whole merge event benefits them. There could be software errors but statistics show that it will not last more than 1 vote period.

PoW chain stranger

There are a lot of unknowns on the PoW chain side. First of all, block time expected to gravitate towards 14 seconds actually could fluctuate a lot due to random nonce evaluation nature (open in new window to see in details):

[

4000×500 31.9 KB

](https://ethresear.ch/uploads/default/original/2X/3/3b4abd5def8726b8bd69071aa36d931af43f3f8f.png)

Here and next we've analyzed the chain on and after the Muir Glacier fork, because behavior is changed after each hard fork.

Looks strange but if we check mean values it's somewhere near expected 14 seconds:

Block time statistics: Mean block time: 13.327266366593754 Deviation: 12.634146977226226 -σ, σ [1 sec, 25 sec]: 85.9% 0, 2σ [0 sec, 38 sec]: 95.0% 0, 3σ [0 sec, 51 sec]: 98.2%

So, actual mean block time is about $$13\frac{1}{3} seconds$$

So, when we calculate block time as 14 seconds, we are getting a growing error.

Another stranger here is difficulty fluctuation. At first let's analyze, for how much will we miss transition start if use proposed compute_transition_total_difficulty at any historical block:

It's not even 0 centered because block time is not 14 seconds usually, let's use 13.33 seconds instead:

Now it's better centered but we see errors of up to 34 hours on historical data. Minus here means we will start transition earlier than expected and plus after expected time.

Also let's estimate the worse possible behavior of Mainnet miners:

- mining power shrinks twice before anchor block
- after anchor block all mining power recovers

If difficulty was allowed to grow in a moment such behavior would reduce transition start delay twice (to 23,502 PoW chain blocks instead of expected 47,003), but as difficulty couldn't grow in a moment, it will actually took at least 23,700 blocks, so difficulty growth cap delays it only for 200 blocks in the worst case.

Discussion

We want the merge to happen in 7 days after the transition start epoch. Our historical data analysis shows that even in happy case this time could be unexpectedly reduced by 1 stalled voting period on Beacon chain (~6.8 hours) because of software errors and ~27.7 hours on PoW chain due to difficulty fluctuation, which is totalled to almost 1.5 days or to be precious, ~34.6 hours. Using block time of 13 1/3 seconds

in calculations reduces it to ~26.3 hours and more correctly guesses real block duration in Mainnet. To exclude fixed point / float calculations, we propose following formula in this case:

SECONDS_PER_ETH1_BLOCK_NUMERATOR = uint64(40) SECONDS_PER_ETH1_BLOCK_DENOMINATOR = uint64(3) def compute_transition_total_difficulty(anchor_pow_block: PowBlock) -> uint256: seconds_per_voting_period = EPOCHS_PER_ETH1_VOTING_PERIOD * SLOTS_PER_EPOCH * SECONDS_PER_SLOT pow_blocks_per_voting_period = seconds_per_voting_period * SECONDS_PER_ETH1_BLOCK_DENOMINATOR // SECONDS_PER_ETH1_BLOCK_NUMERATOR pow_blocks_to_merge = TARGET_SECONDS_TO_MERGE * SECONDS_PER_ETH1_BLOCK_DENOMINATOR // SECONDS_PER_ETH1_BLOCK_NUMERATOR pow_blocks_after_anchor_block = ETH1_FOLLOW_DISTANCE + pow_blocks_per_voting_period + pow_blocks_to_merge anchor_difficulty = max(MIN_ANCHOR_POW_BLOCK_DIFFICULTY, anchor_pow_block.difficulty)

return anchor_pow_block.total_difficulty + anchor_difficulty * pow_blocks_after_anchor_block

In worst scenario we expect 2 consecutive stalled vote periods on Beacon chain and worst hash power changes on PoW chain subtracting respectively 6.8 * 2 = 13.6 hours

and (47,003 - 23,700) * 13.33 seconds = ~86.29 hours

from actual transition period, which could total up to 99.89 hours deduction. So, for Beacon chain nodes in order not to miss expected merge difficulty, all should be ready for terminal Mainnet block in no more than 3 days after transition start epoch in worst scenario with 7 days expected transition setup.