The Second-Slot Itch

Special thanks to Michael Sproul for valuable input.

Blocks are put into slots and can be reorged out of the canonical chain if they are too late and don't get enough attestations. In such cases, the next proposer builds on top of the previous block that has enough attestations. Analyzing the dynamics of reorgs provides valuable insights for CL client teams, helping them to improve their software.

The following covers a brief statistical analysis of reorgs, focusing on CL clients and the second-slot glitch. For more data and visualisations on reorgs, check out reorg.pics.

Dataset - Quick Look

The dataset used in the following focuses on reorged blocks, specifically. I define reorged blocks as slots for which I've seen a valid block in the P2P gossip network that didn't make it to the canonical chain. Missed slots != reorgs, and the following focuses on reorgs:

- epoch

: The epoch number.

- slot

: The unique slot number.

- cl_client

: Client type that experienced the reorg.

- slot_in_epoch

: Position of the slot within an epoch, starting from 0.

The dataset for this analysis includes every single slot/block since the merge.

It contains 1410 reorgs and 17906 slots that were missed by their validators.

Reorged blocks make up 0.059% of the total blocks in that time frame, whereas missed blocks account for 0.757%.

Subsequent slots

Since the merge, there have been 4 reorgs of depth 2. They happened in the slots 5528988, 5920525, 6366357 and 6436520. The rest, 1406 were 1-depth reorgs.

Reorgs:

depth

occasions

example slot(s)

example epoch

1

1406

7060699, 7065797, 7070304

220947, 220806

2

4

5920525, 6366357, 6436520

201141, 198948

Looking at missed slots and reorgs together, we had 323 occasions of consecutive slots with missed/reorged slots/blocks of

depth 2. On the other side, the largest series of missed/reorged slots/blocks is 7 slots long happened on the 11th of May, 2023 and was caused by a bug that made CL clients validate attestation pointing to old beacon blocks (validation requires to recompute an old state or get it from cache - effectively DoSing them), which caused the chain to stop finalising for some epochs.

Reorgs + Missed Slots:

depth

occasions

example slot(s)

example epoch

1

18936

7059904, 7060399, 7060699

220646, 220637

2

323

6994062, 7020298, 7047834

220244, 219384

3

35

6424316, 6753389, 6753462

211045, 211043

4

12

6424242, 6424261, 6753388

211043, 200758

5

5

6424040, 6424205, 6424260

200758, 200756

6

4

6417737, 6424039, 6424204

200756, 200751

7

1

6417736

200554

Looking at clients

The diagram below shows the relative number of reorged blocks per client. For example, 0.1% of the Nimbus blocks were

reorged since the merge. Notably, it appears that blocks built by the Nimbus client got reorged more frequently than other clients. Assuming late blocks are the main reason for getting reorged, this could potentially point to an inefficiency in the Nimbus client causing validators to propose blocks too late in the slot. For Nimbus, an honest reorg strategy that leverages proposer boost to “punish” the previous validator, who accumulated less than 20% of attestations, by reorging them out, might not be the explanation.

[

Screenshot from 2023-08-09 22-18-28

743×281 12.3 KB

](https://ethresear.ch/uploads/default/original/2X/5/59a4782898ec32424d018bda974b53f1aff6eab6.png)

The tricky second slot

The second slot of an epoch tends to be reorged most frequently, followed by occasional reorgs in the third and fourth slots. This phenomenon might stem from the complex epoch boundary posing challenges for the first validator, leading to cascading effects.

[

Screenshot from 2023-08-10 12-34-41

759×259 14.6 KB

](https://ethresear.ch/uploads/default/original/2X/9/92d713536dcdae7b1627d87b10267db7b7a91989.png)

Some statistics later, but it does not take much convincing to realise that there is a significant increase in reorgs in the second and third slot of an epoch. The reasons might potentially

be the following:

- The validators of slot 0 might be delayed due to the computation of the epoch transition, which requires more work compared to other slots within an epoch.
- The validators of slot 1 might try to reorg the block of the slot 0 validators, punishing them for being late (honest reorg).
- Finally, the attesters and the proposer of slot 2 might disagree with the validators of slot 1, reorging them out.
- Eventually, this leads to cascading effects on slot 2 and 3.

In summary, the validator could either be slow and don’t get enough attestations - thus reorged by the next, or the validator tries to reorgs the previous validator and gets reorged by the next one.

Let’s check which client is the most affected. We can do so by plotting the relative probability of a client getting reorged in a specific slot.

[

Screenshot from 2023-08-09 22-54-12

799×273 18.8 KB

](https://ethresear.ch/uploads/default/original/2X/2/21a1558026f231aaf6dfd410fb982da8d2a08c7b.png)

We see that Prysm appears to be the client with the most difficulties with the 2nd slot of an epoch.

Over 0.8% of the Prysm

blocks in the second slot of an epoch were reorged. This is followed by Lighthouse where almost 0.6%

of the 2nd-slot blocks got reorged. Lodestar seems to slightly struggle with the 3rd one.

Diving into Probabilities

In order to understand the potential effects of slot indices within an epoch on the likelihood of reorgs, let’s employ a statistical approach—logistic regression. This method is particularly suited since the target variable, ‘reorged’, is binary, indicating whether a slot has undergone reorg (1) or not (0). By converting the slot_in_epoch

column into dummy variables, each representing a distinct slot index within an epoch, we can identify the effects of individual slots relative to a reference slot (the last one in the dataset).

[

Screenshot from 2023-08-10 13-25-20

931×723 35.2 KB

](https://ethresear.ch/uploads/default/original/2X/0/06be2705ba5ee911e9b4b88e04166abf636ed931.png)

As visible in the output of the logistic regression,

- The coefficient for const

is -7.4625. This is the log odds of reorged

when all predictors are zero, using the last slot in an epoch as the reference category. Simply speaking, this just tells us that the probability of a reorg for the reference category - the last slot in epoch - slightly below the average over all slots at (e^{-7.4625}=0.057\%

)

- The coefficient of the 2nd slot in an epoch (index_of_slot_1

) is 2.3426, suggesting that for a unit increase in this variable, the log odds of reorged

increase by 2.3426, holding all other variables constant. This means that it is 10.41 times (e^{2.3426}=10.41

) more likely to get reorged in the second slot than in the last slot

. A p-value below the commonly used threshold of 0.05 indicates a statistical significance.

- The same applies for index_of_slot_2

with a coefficient of 0.7945. Thus, the chances of being reorged in the third slot are 2.21 higher than in the last slot of an epoch.

- For the 7th slot

in an epoch (index_of_slot_6

) we can see that the probability of being reorged decreases by around 39%

compared to the slots with index_of_slot_31

.

Cascading effects

[

Screenshot from 2023-08-10 13-21-56

745×269 14.2 KB

](https://ethresear.ch/uploads/default/original/2X/1/1852f633e9b871a0eaf3fe12b6cb9911454dbced.png)

1. Constant (Intercept): -7.4279

. This is the log odds of the outcome (reorged

) occurring when prev_reorged

is 0 (i.e., the previous slot did not get reorged). Mathematically, it tells us that the natural log of the odds of a slot getting reorged when the previous slot is not reorged is -7.4279.

1. prev_reorged: 1.5650

. This coefficient represents the log odds ratio and describes how the odds of the outcome (reorged

) change for a one-unit increase in the predictor (prev_reorged

), assuming all other variables in the model are held constant. Given that prev_reorged

is a binary variable (either the previous slot was reorged or it wasn’t), this coefficient tells us the increase in log odds of a

slot getting reorged when the previous slot was reorged compared to when it wasn’t.

To interpret the coefficient in terms of odds:

The odds ratio for prev_reorged

is e^{1.5650} ≈ 4.78

.

This means that when all other predictors are held constant, a slot is about 4.78 times more likely to be reorged

if the previous slot was reorged than if the previous slot was not reorged. Or, to put it another way, there’s a 378% increased odds of a slot being reorged

when the previous slot was reorged compared to when it wasn’t.

Lastly, the p-value for

prev_reorged

is 0.002

, which is less than 0.05, suggesting that this relationship is statistically significant at the 5% level.

Reorgs per client

For the following, the Lighthouse client was used as reference category.

[

Screenshot from 2023-08-10 13-38-22

943×323 17.5 KB

](https://ethresear.ch/uploads/default/original/2X/c/c057cdc97286d77018126a95599365865a0b4c16.png)

- Lodestar

(not statistically significant): * Coefficient: 0.0916. This means, holding all else constant, using Lodestar instead of the reference client is associated with a 1.0958 (e^{0.0916}

) times increase in the odds of reorged

. This is approximately a 9.58% increase in the odds compared to the reference client.

- P-value: 0.785 suggests that the effect of using Lodestar is not statistically significant

at conventional significance levels.

- Coefficient: 0.0916. This means, holding all else constant, using Lodestar instead of the reference client is associated with a 1.0958 (e^{0.0916}

) times increase in the odds of reorged

. This is approximately a 9.58% increase in the odds compared to the reference client.

- P-value: 0.785 suggests that the effect of using Lodestar is not statistically significant

at conventional significance levels.

- Nimbus

(statistically significant): * Coefficient: 0.5593. Using Nimbus, holding all else constant, is associated with a 1.7496 times increase in the odds of reorged

, or a 74.96% increase in the odds compared to the reference client.

- P-value: 0.000 indicates that the effect of using Nimbus is statistically significant

.

- Coefficient: 0.5593. Using Nimbus, holding all else constant, is associated with a 1.7496 times increase in the odds of

reorged

, or a 74.96% increase in the odds compared to the reference client.

- P-value: 0.000 indicates that the effect of using Nimbus is statistically significant

.

- Prysm

(not statistically significant): * Coefficient: 0.0185. Using Prysm, holding all else constant, is associated with a 1.0187 times increase in the odds of reorged

, or approximately a 1.87% increase in the odds compared to the reference client.

- P-value: 0.767 suggests that the effect of using Prysm is not statistically significant.
- Coefficient: 0.0185. Using Prysm, holding all else constant, is associated with a 1.0187 times increase in the odds of reorged

, or approximately a 1.87% increase in the odds compared to the reference client.

- P-value: 0.767 suggests that the effect of using Prysm is not statistically significant.
- Teku

(not statistically significant): * Coefficient: 0.0892. Using Teku, holding all else constant, is associated with a 1.0933 times increase in the odds of reorged

, or a 9.33% increase in the odds compared to the reference client.

- P-value: 0.253 suggests that the effect of using Teku is not statistically significant.
- Coefficient: 0.0892. Using Teku, holding all else constant, is associated with a 1.0933 times increase in the odds of reorged

, or a 9.33% increase in the odds compared to the reference client.

- P-value: 0.253 suggests that the effect of using Teku is not statistically significant.

Wrapping Up

The stats indicate a significant impact of certain slot indices within an epoch on the likelihood of being reorged. Furthermore, there are potential cascading effects of reorgs on consecutive slots. While individual reorg events are quite infrequent, their occurrence has notable effects on the following slot. For the CL clients, it appears there are some inefficiencies causing slots to be reorged.

The code for this analysis is available here.

A special thanks to Michael Sproul for open-sourcing the blockprint tool which helped identify reorgs and affected clients.