(Team's Twitter handles: @0xjorgeth

, @M25Marek

, @rmzcrypt

, @Smartprogrammer.)

Understanding the distribution of Ethereum's execution-layer and consensus-layer clients used by validators is vital to ensure a resilient and diverse network. Although there are currently methods to estimate the Beacon Chain's client distribution among validators, the same cannot be said about execution client distribution. Also, there are no standard means of anonymously showcasing which EL and CL clients are being utilized by validators.

Therefore, as part of the Ethereum Foundation's Data Collection Grants Round 2023 which ran between last September and October, an interdisciplinary team involving Nethermind Research and Nethermind core developers received a grant to work on the project "Allowing validators to provide client information privately". This project aims to research and design a mechanism to submit and extract this crucial data while potentially avoiding compromising user anonymity and network performance. This project aims to research and design a mechanism to submit and extract this crucial data while potentially avoiding compromising user anonymity and network performance. Our work relates to the grants round wishlist items "Using ZK for data collection allows validators to provide information while staying anonymous" and "Improved Validator crawling".

Currently, we see widespread interest in the subject, especially in light of recent client bugs, which have brought client diversity to the forefront of discussion and led our Ethereum community to reflect upon the radically different consequences that a consensus bug in a supermajority execution client would have. As an example of the proposals that have been motivated by this topic, there is currently a PR by @ethDreamer on Ethereum's execution APIs, suggesting some protocol changes to facilitate exposing client data on the graffiti field of a block.

The purpose of this thread will therefore be twofold:

- Share with the community our current impressions on the subject, along with research directions we are thinking of undertaking (and discarding).
- Provide a space for discussion which allows us to keep the community informed on our progress, and welcome feedback throughout all stages of the project.

Objectives

The objectives of our research are as follows:

1. Analyze the technical possibilities for validators to report on (or make visible) EL and CL data, while leveraging existing infrastructure and without compromising performance or security.
2. Identify techniques to anonymize the methods found in Objective 1.
3. Compare the solutions found in Objective 1 and their anonymized versions found in Objective 2 (when applicable) in light of accuracy, security, privacy, and performance concerns.
4. Present our research findings to the Ethereum community and advise on the best option found in the analysis.
5. If the option found in Objective 4 requires protocol changes, prepare an EIP detailing the implementation of this mechanism, incorporating the community's feedback where appropriate.

Background

Our research begins by briefly analyzing the current methods for measuring client diversity in Ethereum, along with their weaknesses. In general, we cluster these methods into three main categories:

1. Use of heuristics or machine-learning tools. This approach is used by Blockprint to "guess the consensus client for a block, based on the similarity of that block to others in its training data." Note that this approach cannot determine the execution client associated with a validator, only the consensus client. This is due to the widespread use of MEV-Boost and the expected transition towards PBS architectures in general, which make it so that there is no fingerprint of the execution client being run by the block proposer in the structure of the block.
2. P2P crawling. A technique used by Miga Labs and Ethernodes; involves running a specialized node (or "crawler") in Ethereum's P2P networks to identify the clients being used by the other nodes. Although we can gather data from both consensus and execution clients, this method suffers from a flaw for our purposes: it finds the client distribution among all Ethereum nodes

in the network, but it does not weigh these nodes according to the number of validators

connected to them. It is the client distribution associated with validators that is important to the health of the Ethereum network, given that these determine consensus.

1. Surveys. By consulting with large node operators and staking pools, we can aim to obtain a large sample of the client distribution of Ethereum validators. Unfortunately, it can be hard to gauge the accuracy and validity of the reported data and whether the utilized sample is representative. This method is used by https://execution-diversity.info, which also powers https://clientdiversity.org/.

Having noted the current room for improvement in these techniques, a natural starting point is building upon extant methods, while brainstorming other possible directions. Approaches to study include but are not limited to:

1. Making client data visible in proposed blocks.

Although we cannot follow Blockprint's heuristic approach to obtain client diversity data for execution clients, we can still source our statistics from block proposers (like Blockprint does) if we explicitly include their client diversity data in the block structure. An example would be pushing this data in the graffiti field during block proposing, which can then be readily collected from the blocks.

1. Improving the reliability of crawling methods.

Are there any acceptable changes to clients that would facilitate the work of crawlers looking to estimate the client distribution amongst validators (not nodes)? An example would be allowing other nodes in the P2P network to gather data on the number of total validators connected to a single node. In this way, crawlers could weigh the contributions of each node to consensus accordingly.

1. Dedicated sub-protocol for client data collection.

Another approach would involve designing a separate sub-protocol for validators to share client data, which would run in parallel to the consensus protocol. This would be a streamlined, client-based alternative to a manual survey, which would aim to achieve better privacy guarantees. Moreover, we aim to isolate this mechanism from Ethereum consensus as much as possible, to avoid any additional demands on speed and performance.

1. Any other approaches found during our research with the potential to lead to a solution will be included as well.

Preliminary analysis

Although our research project has just started, we have internally discussed some of our initial impressions on the aforementioned directions, which are given below. For each direction, we highlight expected protocol changes along with the challenges to solve.

Making client data visible in proposed blocks

A straightforward approach here would be for Ethereum consensus clients to include the consensus/execution client pair being used by a validator at the time of proposing a block, via the graffiti field.

Required protocol changes

- EngineAPI method for the CL client to retrieve the EL client details. We note that there are already discussions on the need to implement this feature.
- (Optional) Clients could agree on an encoding to be used when pushing data to the field, which represents the EL/CL combination. For example, the aforementioned PR by @EthDreamer suggests a two-letter abbreviation for each client.

Challenges

- How should this method deal with multiplexed architectures? For example, we have tools like ExecutionBackup, where multiple EL clients are connecting to the same CL client. Another example is given by tools such as Vouch, which make it hard for a validator to pinpoint a specific CL and EL client being run under the hood. Finally, we can consider the challenge of reports coming from DVT-based validators. It would be advisable to have a way for validators to report when they are running with multiple ELs or CLs concurrently
- How can we add anonymity to this method? Our team is working on this and assessing some ideas to this end.

Improving the reliability of crawling methods

Let us analyze modifications to Ethereum clients so that a crawler can obtain the validator

client diversity distribution, not just the node distribution.

Required protocol changes

- We would require nodes to disclose to crawlers the number of validators connected to them, among other changes.

Challenges

Below, we share some reasons why we believe this is a bad idea:

- It is unclear how this method would handle infrastructure such as load balancers or Distributed Validator Technology. These approaches can have the same group of validators attached to several nodes at once so that crawling the network would result in counting the same validators multiple times.
- Moreover, if users are allowed to self-report the number of validators connected to their node, then they can easily skew the client diversity data by declaring artificially large numbers of validators. Thus, this validator count should be supported by some form of cryptographic evidence, such as attestations or an adaptation of a proof-of-validator.
- There are privacy concerns in disclosing the number of validators attached to a node. This could signal nodes of high importance (when the validator count is high), making them prime targets for a DoS attack.

Due to the above, we are unlikely to suggest pursuing this approach.

Allowing nodes to listen to client diversity data through the gossip network

The analysis above discourages us from pursuing crawlers that use therequest/response domain of the P2P network. Can we leverage Ethereum's gossip domain instead?

We are analyzing the possibility of defining a newGossipSub topic so that validators (or a randomly chosen subset of them at a time) can periodically gossip their client diversity data. In this way, any client in the P2P network can get data by subscribing to this topic.

Required protocol changes

- EngineAPI method for the CL client to retrieve the EL client details, so that the gossiping can take place on the consensus layer entirely.
- The introduction of new GossipSub topics—e.g. client_data_{subset}

—where users can broadcast their client data for it to be aggregated by any interested listener.

- Once more, it would be beneficial for clients to agree on an encoding to represent the EL/CL combination (instead of using plaintext), to minimize the data that is sent.

Challenges

- How can we make sure that the validators publishing data in this topic represent a statistically significant sample of the network? We could, for example, take inspiration from the mechanism of aggregator selection. We could look into gathering this data e.g. only from the validators chosen as part of a given attestation subnet per epoch.
- This would also help limit the rate at which data is gathered to avoid needless overload of the network.
- This would also help limit the rate at which data is gathered to avoid needless overload of the network.
- Gathering this data should impact consensus as little as possible. If the data sample is related to attestation subnets, then we could gossip the data during slots when the validators are not required to gossip their attestations.

There is also the question of how to address privacy and anonymity concerns via the method above, which leads us to the next approach.

Implementing a dedicated private voting scheme for client diversity data

A voting scheme is a digitalized system designed to efficiently vote and count ballots during elections, aiming to streamline the electoral process, enhance accessibility, and minimize errors. In private voting schemes, the primary focus is ensuring maximum confidentiality and anonymity for individual votes. This is achieved by implementing cryptographic techniques like homomorphic encryption and secure multiparty computation, preserving voters' privacy. Private voting systems utilize robust protocols, enabling citizens to cast their ballots electronically while maintaining the secrecy of their choices.

In our case, we are trying to collect client information (as a vote) from the Ethereum users without revealing information about the users. This problem seems to be solved using a private voting scheme if we can deploy a private voting protocol that utilizes Ethereum users as voters. As our first iteration towards using private voting schemes, we look into lightweight protocols that do not require a dedicated blockchain to run, so that they are more adaptable to the P2P context. In this vein, we are analyzing how to adapt the Belenios voting scheme to gather client diversity data.

Required protocol changes

Various cryptographic building blocks and structures would need to be implemented. Such a protocol will require, among other things:

- A distributed key generation protocol amongst the parties that are chosen as decryption authorities.
- The ability for validators to encrypt their votes via a homomorphic encryption scheme (e.g. El-Gamal).
- The infrastructure for voters to generate zk-proofs of their votes' validity.

Challenges

- Which parties are best suited to play the role of decryption authorities? Note that they can prevent the voting results from being known if enough of them decide not to participate.
- Is checking the validity of the votes a bottleneck? Can this protocol be realistically executed without putting excessive computing/networking demands on validators? Additional analysis of the proving times and message sizes is currently underway.

Towards a rubric for assessing the best solution

As we develop our ideas into concrete approaches, we will need a rubric to assess the merits of each solution, as part of Objective 3. Here is a preliminary rubric that encompasses various points relevant to the problem:

- Complexity: how many changes to the protocol/clients are required? Ideally, we want solutions that are not overly invasive with regards to modifying key components of the Ethereum protocol.
- Privacy: can the solution be made private, to accommodate validators that do not wish their data to be publicly known?
- Reliability: how easy it is to submit fake data for a given approach? At the end of the day, a sufficiently motivated party should be able to turn off or bypass these data-gathering features. (We hope to address this by building a highly functional solution where the incentives for doing this are scarce)
- Sampling rate/data density: how long does it take to gather a meaningful statistic? We aim to strike a balance here, such that data can be gathered sufficiently quickly, but without meaninglessly overloading the protocol.

This marks the end of our kickoff update. Our team looks forward to carrying on with this research project and sharing any updates under this post.