Preliminary Ethereum 2.0 Client Metrics for Early Benchmarking

Afri Schoedon, @q9f

June 25, 2020

I. Introduction

Ethereum 2.0 will be a new blockchain protocol enabling – amongst others – horizontal scalability through sharding and transitioning the chain to a proof-of-stake consensus algorithm.

None of the features that Ethereum 2.0 will bring are being implemented in established Ethereum 1.x clients such as Geth or Besu. Therefore, it's being worked on a new generation of core clients to power the beacon chain. None of these clients has ever been used in production before.

With the launch of the beacon chain supposedly happening in 2020, a first gathering of key metrics of three selected Ethereum 2.0 clients will be conducted, namely Lighthouse, Prysm, and Teku.

This article seeks to document the gathered metrics of different clients adhering to scientific methodology. It does not, however, intend to replace a peer-reviewed publication. It's simply a version of the data commented by the author, enriched with emojis. ;)

The raw data is available on Github[1] for further analysis.

II. Clients

Three clients are used for comparing key-performance metrics.

Lighthouse is an Ethereum 2.0 client developed by Sigma Prime[2]. It's implemented in the Rust programming language. Data referring to the Lighthouse client is depicted in orange throughout this document (figure 1).

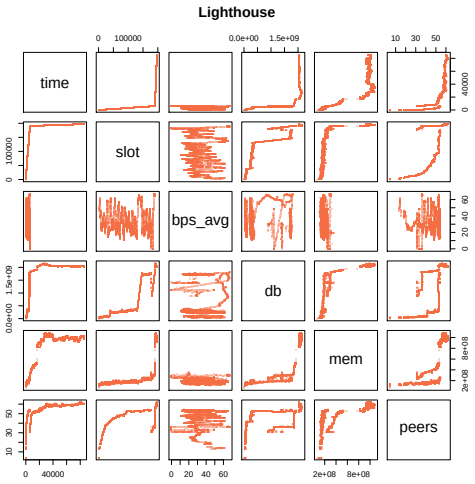

Figure 1: *Lighthouse is depicted in orange. All data collected is displayed in this matrix: time running, slot height, blocks per second, database size, memory usage, and peer count.*

Prysm is a beacon-chain implementation written in Go[3]. It's being maintained by the Prysmatic Labs team. Data referring to the Prysm client is depicted in purple throughout this document (figure 2).

Teku is an enterprise-grade Ethereum 2.0 client built by the PegaSys Engineering team[4]. It's implemented in Java and data referring to the Teku client is depicted in turquoise throughout this document (figure 3).

Other clients implementing the Ethereum 2.0 protocol exist, namely ChainSafe Systems'

[1] github.com/q9f/eth2-bench-2020-06
[2] github.com/sigp/lighthouse
[3] github.com/prysmaticlabs/prysm
[4] github.com/PegaSysEng/teku

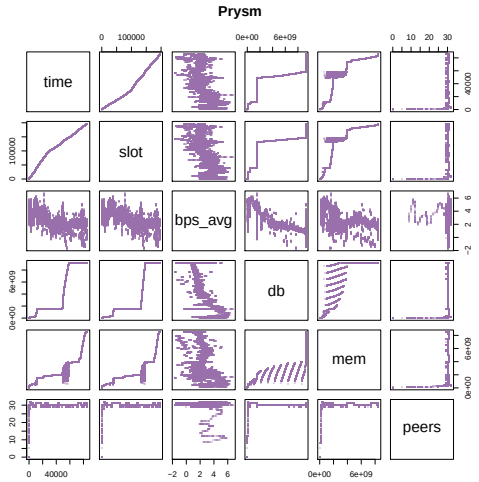

Figure 2: *Prysm is depicted in purple. All data collected is displayed in this matrix, same as figure 1.*

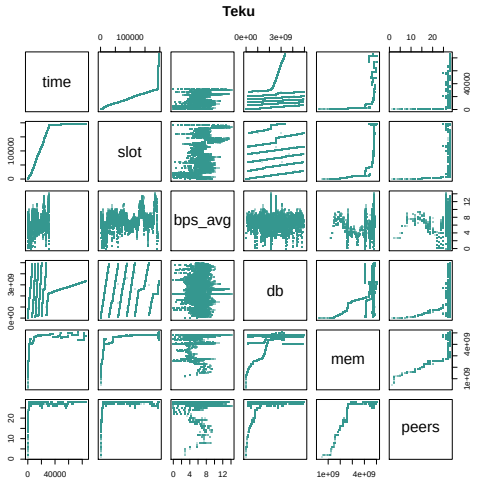

Figure 3: *Teku is depicted in turquoise. All data collected is displayed in this matrix, same as figure 1.*

Lodestar[5], Status' Nimbus[6], Nethermind's Cortex[7], and the Ethereum Foundation's very own Trinity[8]. Due to the different progress of implementing the protocol specification and core components, these clients were not considered for comparison, yet.

III. Metadata

The data is gathered on the Witti testnet[9]. Witti is the second multi-client testnet launched with the three in section II introduced clients as genesis validators.

At the time of collecting the metrics, the Witti testnet is based on `v0.11.3` of the Ethereum 2.0 beacon-chain specification. It contains approximately 190,000 slots and is run by 989 validators.

i. Host Systems

Three identical host systems have been installed for the sole purpose of the performance inspection. The host systems are virtual private servers with an Arch Linux operating system kernel version `5.7.2-arch1-1`.

The host machines are powered by an AMD EPYC 7281 CPU with 4 virtual cores. The available memory is 12 GiB and the NVMe disk allows for 120GiB capacity.

It shall be noted, that these machines are not high-performance machines. This is intentional. The author subjectively tries to create an environment closer to real-world conditions. It is assumed that many users run their beacon-chain nodes *at home*.

Furthermore, it shall be emphasized, that this is a preliminary, high-level benchmark. It should be repeated in the future on more appropriate bare-metal machines after stable client-releases are available to eliminate potential performance invariance caused by virtualized systems[10].

ii. Client Versions

All clients were compiled on June 22nd, 2020, from the latest available source-code targeting

[5] github.com/ChainSafe/lodestar
[6] github.com/status-im/nim-beacon-chain
[7] github.com/NethermindEth/cortex
[8] github.com/ethereum/trinity
[9] github.com/goerli/witti

[10] Careful readers will find competition for resources of the three virtualized host systems apparently sharing one physical host in figure 5.

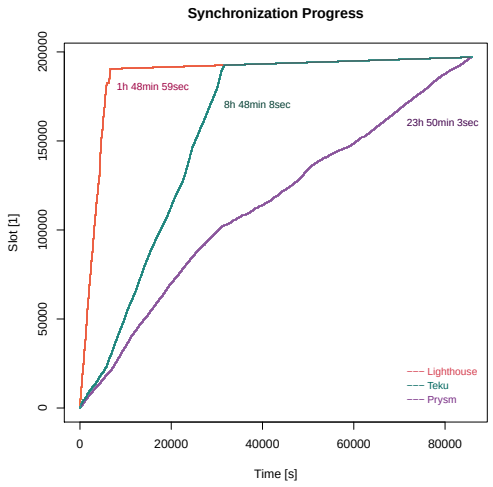

Figure 4: *Synchronization progress over time.*

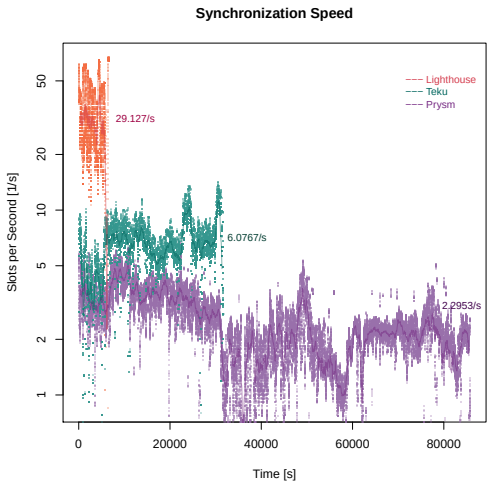

Figure 5: *Synchronization speed over time.*

the version `v0.11.3` of the Ethereum 2.0 specification.

- **Lighthouse**: version `lighthouse/0.1.2`, compiled from `master` branch at commit `710409c2b` from June 20th, 2020, with Rust version `1.44.1` stable through Cargo.
- **Prysm**: compiled from `witti` branch at commit `172435374` from June 7th, 2020, with Go version `1.14.4` through Bazel.
- **Teku**: version `teku/v0.11.5-dev`, compiled from `master` branch at commit `3bd726809` from June 22nd, 2020, with Java version `14.0.1` through Gradle.

All clients were provided with a sufficient number of bootstrap nodes to ensure good connectivity and eliminate potential networking bottlenecks (compare section IV point iv).

IV. Performance

This document does only inspect the performance metrics of beacon-chain node implementations. Other features such as running validator clients, bootstrap nodes, or other relevant tooling is disregarded for simplicity.

i. Synchronization Metrics :)

Starting this section with an emoji to relax the tension – synchronization metrics often lead to heated debates on the world wide web.

Figure 4 displays the progress of synchronizing the three aforementioned clients. Notably, the Lighthouse client manages to fully synchronize all blocks and verify all signatures in a little less than 1 hour and 49 minutes. Teku completes the same task in 8 hours and 48 minutes, whereas Prysm requires 23 hours and 50 minutes to fully sync and verify the Witti beacon chain.

In addition, figure 5 displays the same data but computing the synchronization speed in slots per second by taking the time required to fully catch up with the beacon-chain head. The plotted data points display a moving average over 60 seconds, the plotted line shows a moving average over 10 minutes. Lighthouse appears to be roughly one order of magnitude faster than Teku and Prysm.

The data at glance.

- **Lighthouse** synchronizes 190,463 slots in 6,539 seconds at an overall average speed of 29.127 slots per second.
- **Teku** synchronizes 192,560 slots in 31,688 seconds at an average speed of 6.0767 slots

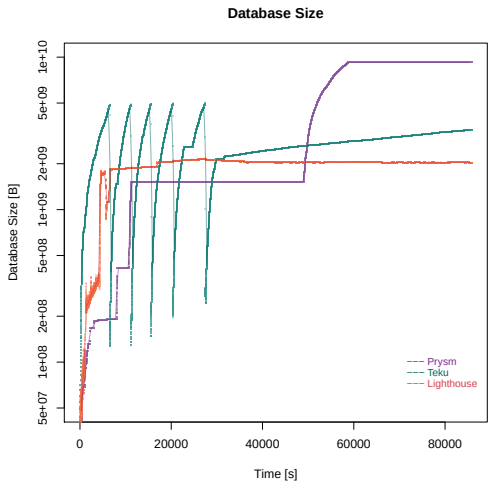

Figure 6: *Database size over time.*

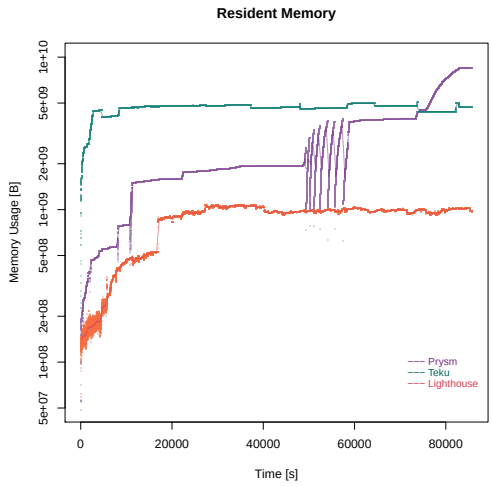

Figure 7: *Resident memory usage over time.*

per second.
- **Prysm** catches up with 196,943 slots in 85,803 seconds at 2.2953 slots per second.

It's important to note that both Teku and Lighthouse do a full verification of all signatures during synchronization *by default* whereas Prysm requires the `-initial-sync-verify-all-signatures` flag to enable full sync.

Assuming a year has 365 days, and the beacon-chain maintains a slot time of 12 seconds, we would see 2,628,000 slots per year. A beacon-chain running under the same conditions as the Witti testnet used for gathering these metrics would require the following synchronization times.

- **Lighthouse** would require approximately 25 hours to synchronize a year's worth of slots on the Witti testnet.
- **Teku** would require around 5 days to synchronize the 2,628,000 slots.
- **Prysm** would require circa 13 days for the same task.

These numbers are highly hypothetical and should not be taken out of context. Due to finality and weak subjectivity in Ethereum 2.0's proof-of-stake, end-user client synchronization strategies might look very different from what we have experienced in Ethereum 1.x, e.g., by fetching a finalized state and syncing from there.

ii. Database Metrics

Figure 6 displays the database size in Bytes plotted over time of running the nodes. The patterns are left uncommented for the client developers to analyze.

The data at glance.

- **Lighthouse** would require 1.89 GiB for 197,068 slots worth of beacon-chain data.
- **Teku** would require 3.11 GiB for 197,068 slots.
- **Prysm** would require 8.59 GiB for 196,943 slots.

Assuming linear growth of the beacon-chain client's databases, we would look at the following yearly growth rates for each client under the conditions comparable to the Witti testnet.

- **Lighthouse** would require 25.8 GiB database space for a year's worth of beacon-chain data.
- **Teku** would require 42.6 GiB per year.
- **Prysm** would require 117.5 GiB every year under these conditions.

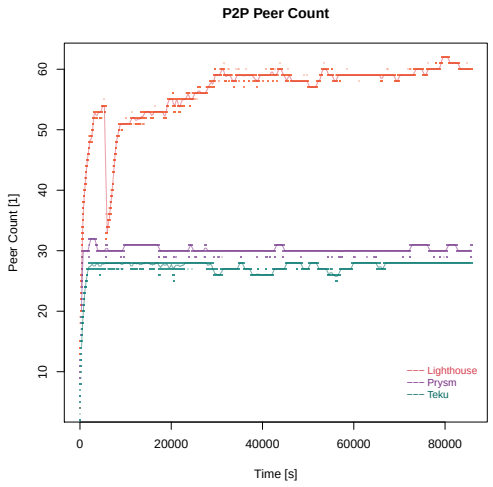

Figure 8: *Client's peer count over time.*

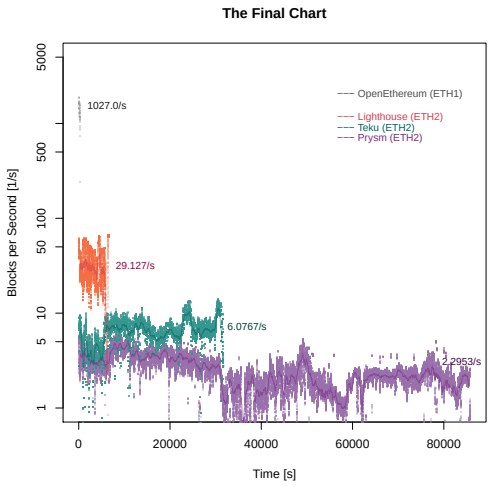

Figure 9: *Adding an Ethereum 1.x client to the plot.*

However, due to the safe pruning of historic states and advanced database configuration options[11], linear growth is not to be expected for end-user clients. Therefore, these data points should be regarded very carefully and only serve as additional context.

iii. Memory Metrics

Figure 7 displays resident set size reported by the three clients. Again, the patterns are left uncommented. Notably, the Lighthouse client appears to be most efficient with regard to memory usage, peaking at around 1 GiB in default operation mode. Teku peaks at just below 5 GiB, Prysm at a little less than 8 GiB.

iv. Networking Metrics

Figure 8 displays the peer count of every client during operation. There is not much to be commented on and just serves as a sanity check to rule out networking issues that could impact any of the other metrics.

Notably, the drastic drop in peers of the Lighthouse client might be related to a bug that's currently being investigated[12].

V. Apples and Oranges

Now, to round this up, an Ethereum 1.x protocol provider will be added to the mix.

Figure 9 adds some context to the synchronization speed depicted in figure 5. In addition to the data points of Lighthouse, Teku, and Prysm, an OpenEthereum[13] node is added in black. OpenEthereum is an Ethereum 1.x client implementation maintained by Gnosis.

The client was tasked to synchronize and fully verify the first 302,967 blocks on the Goerli testnet[14]. It completed in 295 seconds with an overall average synchronization speed of 1,207 blocks per second making it two orders of magnitude faster than Lighthouse, or even three orders of magnitude faster compared to Teku and Prysm.

Furthermore, it managed to synchronize the entire active blockchain of the testnet with 2,924,236 blocks in 4 hours and 15 minutes at an overall average speed of 191.3 blocks per second.

[11] lighthouse-book.sigmaprime.io/advanced_database
[12] github.com/sigp/lighthouse/issues/1272
[13] github.com/openethereum/openethereum
[14] github.com/goerli/testnet

VI. Conclusion

Even though it's early to make any substantial conclusions and the clients are still undergoing very active development, two things cannot be ignored when looking at the data.

First of all, it's difficult to compare Ethereum 1.x clients with Ethereum 2.0 clients due to the entirely different architecture and cryptography they require. However, from a practical perspective, these numbers *do* matter for some – if not many – users out there. While being fully aware of the ongoing active development in the Ethereum 2.0 client landscape, we should be encouraged to double down research on increasing these numbers by *orders of magnitude* rather than *some percentage*.

Being able to verify a block in 10 milliseconds or less (hypothetical number) is not only faster and more convenient for users but also improves the node's stability and the robustness of the network against denial-of-service attacks.

Secondly, a huge shoutout to the Sigma Prime team positioning their Lighthouse client on top in every metric collected. It shall be noted that Ethereum 2.0 is not a race or competition but rather a massive collaborative effort to design and implement a new blockchain protocol. While this is not about calling out a winner, we should be encouraged to learn from the Sigma Prime team's design decisions and the Lighthouse architecture to potentially improve other Ethereum 2.0 node implementations.

Figure 10: *Apples and Oranges.*

Note

The author is not affiliated with any of the teams implementing an Ethereum 2.0 client. The author is independently funded through the Ethereum Foundation's Ecosystem Support Program[15] and the Goerli Testnet Initiative[16].

The author is not speaking on behalf of any organization.

Last but not least, the author would like to thank Ronin Kaizen and Franzi Heintel for checking this document for its integrity and Danny Ryan for giving this data a quick review as well as enough context to rule out common pitfalls analyzing Ethereum 2.0 specification implementations.

Thank you.

:)

[15] esp.ethereum.foundation

[16] goerli.net