Ethereum 2.0 Client Metrics

Afri Schoedon, @q9f

July 17, 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.

i. Motivation

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, the second gathering of key metrics of four selected Ethereum 2.0 clients will be conducted, namely Lighthouse, Prysm, Teku, and Nimbus.

This work shall allow insights into the performance and stability of the given beacon-chain node implementations.

ii. Previous Benchmark

In June 2020, a similar, preliminary benchmark has been conducted[1] gathering first insights into client metrics and getting feedback from the Ethereum 2.0 core-developer community.

Before diving into the results, please note the following.

- Most importantly, this work adds Nimbus to the list of profiled clients, allowing for comparing four client's metrics instead of three.
- The numbers in this report are *not* comparable with numbers in the previous report. This is mainly due to using different, dedicated, bare-metal hardware for gathering these numbers as compared to the virtual hosts used in the previous work.
- Unfortunately, the previous report contained a methodological inaccuracy. While all clients were run under the same conditions doing a full synchronization, the Prysm client was not built with optimized compiler settings. The team is aware and the documentation will be updated accordingly for all users[2]. This has been revised and all clients are provided with release binaries.
- Last but not least, this benchmark was conducted on the Altona testnet[3]. In contrast to the Witti testnet used in the last report, the Altona testnet has a different composition of validators and currently contains fewer blocks than the Witti testnet had in June 2020.

iii. Commented Data

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.

The raw data is available on Github[4] for further analysis.

II. Clients

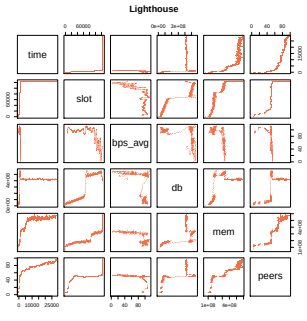
Four clients are used for comparing key-performance metrics.

[1] github.com/q9f/eth2-bench-2020-06

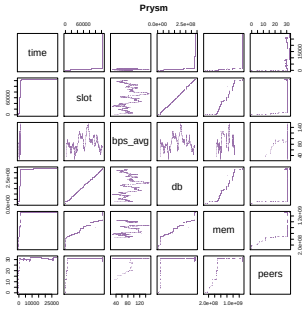
[2] prysmaticlabs/documentation#189

[3] github.com/goerli/altona

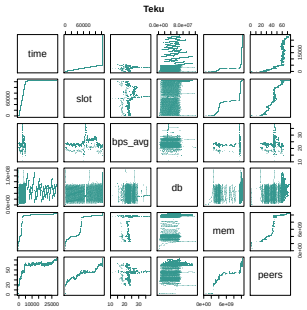
[4] github.com/q9f/eth2-bench-2020-07



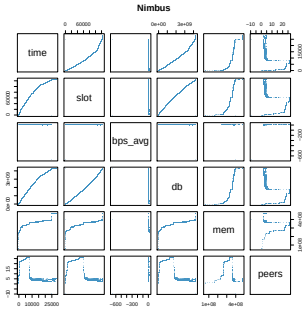
(a) *Lighthouse is depicted in orange.*



(b) *Prysm is depicted in purple.*



(c) *Teku is depicted in turquoise.*



(d) *Nimbus is depicted in blue.*

Figure 1: *All data collected is displayed in these matrices: time running, slot height, blocks per second, database size, memory usage, and peer count.*

Lighthouse is an Ethereum 2.0 client developed by Sigma Prime[5]. It's implemented in the Rust programming language. Data referring to the Lighthouse client is depicted in orange throughout this document (figure 1a).

Prysm is a beacon-chain implementation written in Go[6]. It's being maintained by the Prysmatic Labs team. Data referring to the Prysm client is depicted in purple throughout this document (figure 1b).

Teku is an enterprise-grade Ethereum 2.0 client built by the PegaSys Engineering team[7]. It's implemented in Java and data referring to the Teku client is depicted in turquoise throughout this document (figure 1c).

Nimbus is a beacon node implementation written in Nim built by the Status team[8]. Data referring to the Nim Beacon Chain client is depicted in blue throughout this document (figure 1d).

Other clients implementing the Ethereum 2.0 protocol exist, namely ChainSafe Systems' Lodestar[9], Nethermind's Cortex[10], and the Ethereum Foundation's Trinity[11]. 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 Altona testnet. Altona is the third multi-client testnet launched with the four in section II introduced clients as genesis validators.

At the time of collecting the metrics, the Altona testnet is based on `v0.12.1` of the Ethereum 2.0 beacon-chain specification. It contains approximately 120,000 slots and is run by 3,792 validators.

[5] github.com/sigp/lighthouse
[6] github.com/prysmaticlabs/prysm
[7] github.com/PegaSysEng/teku
[8] github.com/status-im/nim-beacon-chain
[9] github.com/ChainSafe/lodestar
[10] github.com/NethermindEth/cortex
[11] github.com/ethereum/trinity

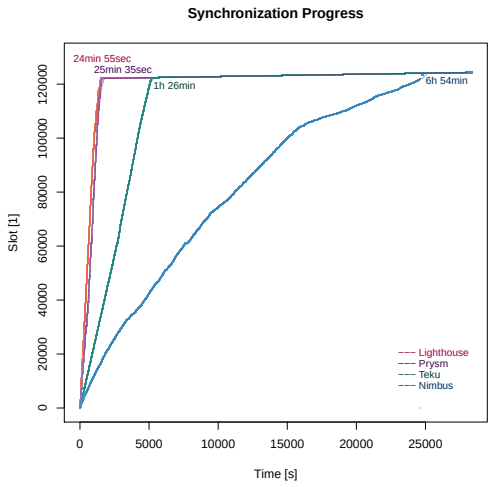


Figure 2: *Synchronization progress over time.*

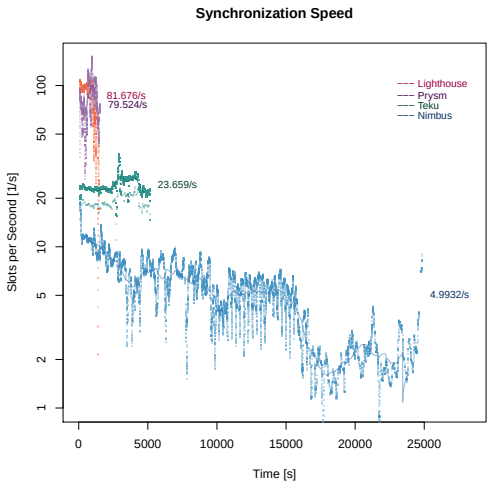


Figure 3: *Synchronization speed over time.*

i. Host Systems

Four identical host systems have been installed for the sole purpose of the performance inspection. The host systems are dedicated bare-metal servers with an Ubuntu 20.04 LTS operating system kernel version `5.4.0-40-generic`.

The host machines are powered by an Intel Xeon E3-1240 v6 CPU with 8 cores. The available memory is 32 GB and the SSD disks allow for 250GB capacity.

ii. Client Versions

All clients were compiled on July 16th, 2020, from the latest available source-code targeting the version `v0.12.1` of the Ethereum 2.0 specification.

- **Lighthouse**: version `lighthouse/0.1.2`, compiled from `master` branch at commit `fc5e6cbb` from July 16th, 2020, with Rust version `1.44.1`-stable through Cargo.
- **Prysm**: compiled from `master` branch at commit `df738517` from July 16th, 2020, with Go version `1.13.8` through Bazel.
- **Teku**: version `teku/v0.12.2-dev`, compiled from `master` branch at commit `04b0a00a` from July 16th, 2020, with Java version `14.0.1` through Gradle.
- **Nimbus**: version `beacon_node v0.5.0`, compiled from `devel` branch at commit `3dfbc311` from July 15th, 2020, with Nim version `1.2.2` through Make.

All clients contain a built-in Altona configuration and 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 only inspects the performance metrics of beacon-chain node implementations. Other features such as running validator clients, bootstrap nodes, or other relevant tooling are disregarded for simplicity.

i. Synchronization Metrics

Figure 2 displays the progress of synchronizing the four aforementioned clients. Notably, the Lighthouse client manages to fully synchronize all blocks and verify all signatures in a little less than 25 minutes with the Prysm client being *on par* finishing the same task in just about the same time. Teku completes the same task in 1 hour and 26 minutes, whereas Nimbus

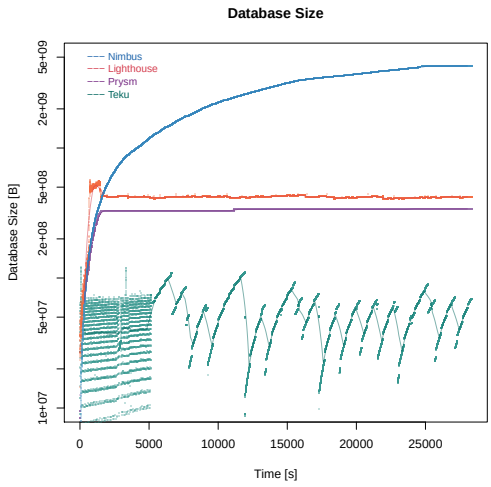


Figure 4: *Database size over time.*

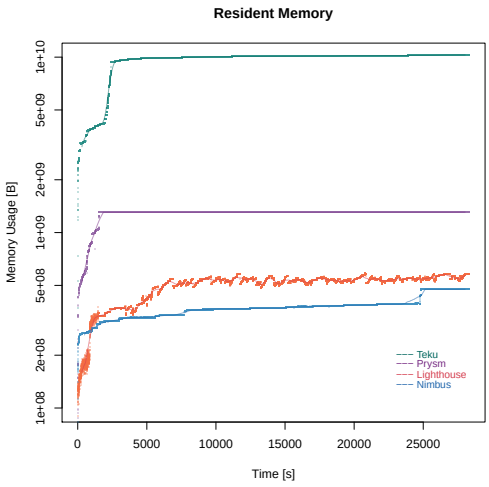


Figure 5: *Resident memory usage over time.*

requires 6 hours and 54 minutes to fully sync and verify the Altona beacon chain.

Also, figure 3 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 and Prysm lead the chart at an average of approximately 80 slots per second on the dedicated hardware.

The data at glance.

- **Lighthouse** synchronizes 122,105 slots in 1,495 seconds at an overall average speed of 81.676 slots per second.
- **Prysm** catches up with 122,069 slots in 1,535 seconds at 79.524 slots per second.
- **Teku** synchronizes 122,412 slots in 5,174 seconds at an average speed of 23.659 slots per second.
- **Nimbus** catches up with 124,051 slots in 24,844 seconds at 4.9932 slots per second.

All clients do a full verification of all signatures during synchronization by default. The teams currently working on integrating new `bls` libraries.

ii. Database Metrics

Figure 4 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.

- **Teku** requires 66.3 MiB after 124,342 slots.
- **Prysm** requires 324 MiB after 124,342 slots.
- **Lighthouse** requires 403 MiB after 124,342 slots.
- **Nimbus** requires 3.98 GiB after 124,295 slots.

The data indicates that the Teku, Prysm, and Lighthouse clients implement database pruning, i.e., by removing everything that is invalidated by finalization or non-checkpointed states. The Nimbus client is running in archive mode by default.

iii. Memory Metrics

Figure 5 displays resident set size reported by the four clients. Again, the patterns are left uncommented. Notably, the Nimbus and Lighthouse clients appear to be most efficient concerning memory usage, requiring around 500 MiB in default operation mode. Prysm peaks at just below 1.3 GiB.

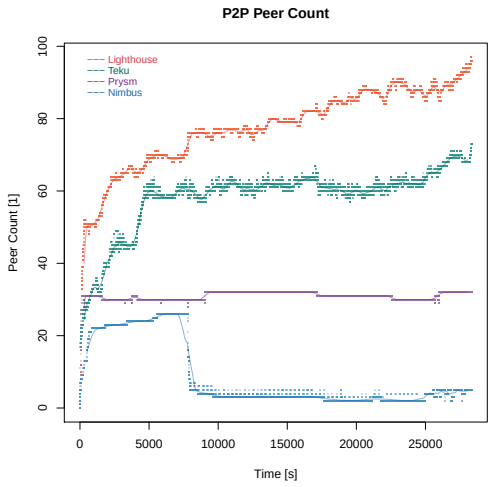


Figure 6: *Client's peer count over time.*

Teku reports a little less than 10 GiB. The actual Java heap memory used by Teku on Altona can be assumed much lower. The off-heap memory that Java allocates is outside of the team's easy control. The JVM is being greedy about available memory, however, it is still possible to run Teku nodes on machines with very small available memory, e.g., 2GB.

iv. Networking Metrics

Figure 6 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, there is a drastic drop in peers of the Nimbus client which, however, does not appear to correlate to any of the other metrics collected above.

V. Conclusion

The plots allow for an overview of key performance and stability metrics of the four tested clients.

Notably, both Lighthouse and Prysm appear to be highly optimized in their performance and mature in the implementation of the beacon-chain specification.

The relatively new Teku client already shows good performance but the metrics allow the conclusion that there is still room for optimization, especially regarding its memory footprint.

The Nimbus client which premiered the first time as genesis validator on the Altona testnet shows potential for implementing further features such as pruning and optimizations of the networking and verification code.

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[12] and the Goerli Testnet Initiative[13].

The author is not speaking on behalf of any organization.

A warm note of thanks goes out to everyone who reviewed the initial June-2020 report and provided valuable feedback allowing for a more accurate data gathering in this subsequent report.

And finally, a big thanks to the client teams patiently answering questions and sharing insights about the protocol implementations.

:)

[12] esp.ethereum.foundation
[13] goerli.net