Received 12 February 2024, accepted 16 March 2024, date of publication 25 March 2024, date of current version 29 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3381782

RESEARCH ARTICLE

Can We Run Our Ethereum Nodes at Home?

MIKEL CORTES-GOICOECHEA[1], TARUN MOHANDAS-DARYANANI[1], JOSE LUIS MUÑOZ-TAPIA[2], AND LEONARDO BAUTISTA-GOMEZ[3]
[1]Barcelona Supercomputing Center, 08034 Barcelona, Spain
[2]Universidad Politécnica de Catalunya, 08024 Barcelona, Spain
[3]Status.im & MigaLabs, 08004 Barcelona, Spain

Corresponding author: Mikel Cortes-Goicoechea (mikel.cortes@bsc.es)

This work was supported in part by the Lido Ecosystem Grant Organization (LEGO), in part by Spanish TCO-RISEBLOCK Project under Grant PID2019-110224RB-I00, in part by the Ethereum Foundation under the Research Grant FY21-0356, and in part by Protocol Labs under its Ph.D. Fellowship Program under Grant FY22-P2P.

ABSTRACT Scalability is a common issue among the most used permissionless blockchains, and several approaches have been proposed to solve it. However, tackling scalability while preserving the security and decentralization of the network is a significant challenge. To deliver effective scaling solutions, Ethereum achieved a significant protocol improvement, including a change in the consensus mechanism towards Proof of Stake. This improvement greatly reduced the hardware requirements to run a node, leading to significant sustainability benefits with a lower network energy consumption. This work analyzes the resource usage behaviour of different clients running as Ethereum consensus nodes, comparing their performance under different configurations and analyzing their differences. Our results show higher requirements than initially claimed and show how different clients react to network perturbations. Furthermore, we discuss the differences between the consensus clients, including their strong points and limitations.

INDEX TERMS Ethereum, consensus clients, hardware requirements, proof of stake.

I. INTRODUCTION

Ethereum [1] has been a remarkable achievement on the road to ubiquitous blockchain technology. It led to considerable growth in decentralized applications and the Web3 space due to its pioneer multipurpose Ethereum Virtual Machine (EVM) [2] and its dedicated programming language Solidity [3]. With an extended network of more than 10.000 active nodes [4] and its capabilities to process *Smart Contracts*, it currently handles above 1 million transactions per day from 600.000 active accounts [5]. These characteristics have set the conditions for a solid community of developers and continuous advancements, as well as introducing new technological possibilities. As the adoption of Ethereum increases, its usability is threatened by rising transaction volume and network clogging.

Following the highly tested consensus mechanism at the moment, initially, Ethereum relied on Proof of Work (PoW) to reach consensus [6] among its participants. However, as the very first blockchain ever presented to the world, Bitcoin [7], Ethereum shared a similar set of limitations in terms of scalability and sustainability [8].

In an attempt to make the network more sustainable while presenting a solid foundation to scale up the network capabilities, in 2020, Ethereum embraced a live consensus transition towards a more sustainable Proof of Stake (PoS) [9] based on Casper FFG for finalization [10] and LMD GHOST [11] as a fork-choice rule. With this transition, network participants no longer had to compete with each other to be the first ones to find a block with a valid hash, as the block proposers are randomly selected from the set of active validators using the RANDAO[1] algorithm.

Due to the complexity of introducing this consensus mechanism, the transition was divided into different phases, where all the changes were gradually applied and tested before the final merge happened. In the first phase, in December 2020, PoS was implemented on the Beacon Chain, a live and independent blockchain running in parallel to Ethereum's

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

[1]https://github.com/randao/randao

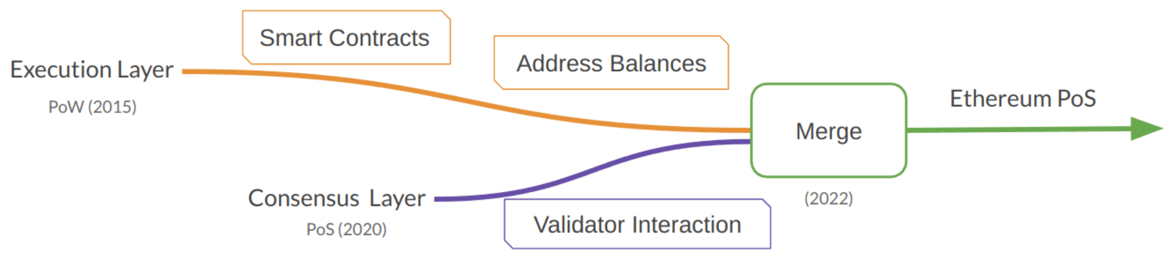

FIGURE 1. Ethereum transition from PoW to PoS.

PoW main network. To participate in Ethereum's PoS consensus as a validator, users or entities must deposit the Ethereum PoS's smart contract [12], activating new validators into the chain.

In the beacon chain, only active validators can participate in block proposals and attestation committees, assuming the duty and responsibility to participate in the consensus over each proposed block. Based on the previous state of the chain and the balances, a randomly chosen validator has a time window of 12 seconds to propose a new block. This time window, or slot, also represents the time range other committee participants had to perform their attestations (visually represented in Figure 2). If a validator does not perform its duties, it gets exposed to economic penalties, while honesty and participation get rewarded. The current PoS model requires the participation of 2/3 of the active validators to consider previous blocks finalized (immutable, as they achieve enough support from the validators). In fact, the network organizes the slots in epochs, where each epoch includes 32 slots.

At the first stage of the transition, the only link between both blockchains was the activation smart contract. But later on, with the Merge on September 15th, 2023, both chains merged into a single one (see Figure 1). However, as Figure 3 shows, the blockchain keeps two distinct layers: i) the execution one and ii) the consensus one, remaining with one blockchain, one block on each slot, and two layers underneath. The Execution Layer (EL) works like the previous Ethereum network, where users broadcast transactions while interacting with the EVM. On the other hand, the Consensus Layer (CL) keeps track of the duties, balances, state, and performance of each validator in the beacon chain.

Each layer has specific requirements and has specific software or clients to participate in. However, each layer's clients can't work alone without a direct pair from the other layer, creating a tandem. Even though the execution layer node could receive the transactions over the network, it won't certainly know which is the last finalized slot that defines the balance at a given Ethereum address. In the same way, if the consensus client doesn't have a connection with the execution layer client, it won't be able to validate the transactions from any proposed block or even fill up blocks with valid transactions if a hosted active validator becomes a block proposer. On top of that, if a user or the entity wants to contribute to the chain's consensus, they will have to plug a third client on top of the beacon node (consensus layer client), a validator client. This third client will be responsible for signing the validator duties for each validator it hosts, as it can run thousands of validators under the same beacon node. The idea behind running a third software on top of the beacon node is to preserve the anonymity of validators in the peer-to-peer network, which makes targeted attacks harder as there is no direct link between which beacon node hosts which validators.

In the original track of the beacon chain[2], there was a second underlined solution to tackle one of the most common limitations of blockchain protocols: the protocol's scalability. The original idea of increasing Ethereum's scalability was adding blockchain shards on the consensus layer, where the whole list of active validators could split across the available shards and limit their activity to the same one. This sharding schema was postponed due to the large complexity of the model. However, thanks to the current stability of the network after the merge, new sharding alternatives have been proposed, i.e., data sharding [13], [14].

Ethereum's approach to updating its consensus mechanism starts from the major shortcomings of the previous PoW. PoW is known for being a resource-intensive consensus mechanism [15], where users need to invest significant amounts of money on hardware to increase their chances of being a block proposer, thus being more profitable. However, this also means that, as the hardware improves the mining capabilities over time [16], the more obsolete the previous hardware becomes. This highly incentivizes the acquisition of newer mining devices, indirectly increasing the network's overall energy consumption [17] over time.

Ethereum's transition to PoS has been highly motivated by this drawback. Although you still need to invest money to be profitable (you need 32 Ether to activate a validator), the hardware requirements to run multiple validators barely change from the node's perspective, as a single beacon node can host up to thousands of validators. The motivation of the community and the core developers is clear: try to reduce the requirements to run a node so that it becomes more accessible, improving the sustainability aspect and the resilience of the network, as having a full Ethereum node at home became less restrictive (from an infrastructure point of view).

[2]https://notes.ethereum.org@vbuterin/SkeyEI3xv

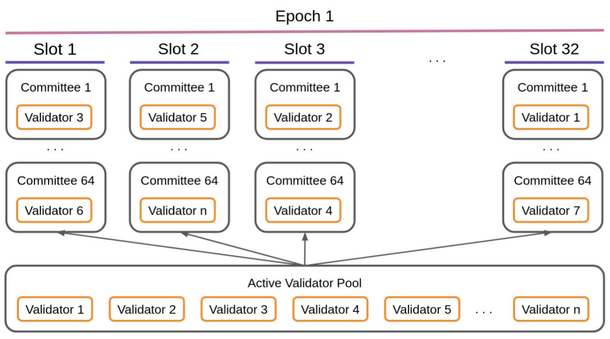

FIGURE 2. Assignation of active validators into attestation committees.

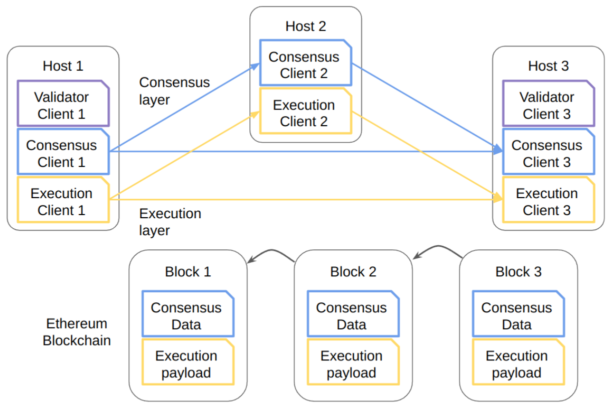

FIGURE 3. Subdivision in layers of Ethereum after the merge.

This paper thoroughly introduces the hardware requirements that the main beacon chain clients need over the different stages of their lifetime and over multiple hardware combinations. We empirically demonstrate and quantify the minimum resources needed to run a full Ethereum node after the merge, and we extensively associate the relation of each event in the network with the specific hardware it requires. By doing this, we present i) the impact of the different approaches each client developer team made into the user's hardware, ii) the importance of these measurements to spot bugs and misbehaving clients under specific network states, iii) the possibility of understanding the healthiness of the chain in past periods only by looking at the synchronization of the chain at that particular time range, iv) the evolution of this requirements as the chain keeps operating. We want to mention that this work has been carried out in close collaboration with the Ethereum Foundation and the client core developer teams. Some of these teams have benefited from this collaboration by identifying bottlenecks and implementing some of the improvements raised by this study. The contributions of this study have also been considered attractive by the Ethereum Foundation, which has asked to perform it continuously, offering a real-time dashboard that can help future users understand which client must be the best fit for their needs.

The remainder of this paper is organized as follows. Section II discusses related work. Section III explains the methodology used for the evaluation. In Section IV, we introduce the measured hardware resource the clients need on different hardware platforms. Section V presents the correlations and insights of the study. Section VI discusses the paper's main findings. Finally, Section VII concludes this work and presents some future directions.

II. RELATED WORK

PoW's security comes with the aggregation of more hashing power to the network since more people honestly participating means more resources an attacker has to invest to achieve 51% of the mining power [18]. However, this is a double-edged sword, as the hardware requirements and the energy consumption increase as more users participate in the PoW consensus mining.

This was clear first in Bitcoin and later on in Ethereum, where the total hashing power of the network has been constantly increasing over time [19], [20], [21]. Lots of efforts in the research and industrial communities were dedicated to optimising and speeding up the process of hashing [22], [23]. Furthermore, the direct application of those hashing methodologies on mining [24] defined a rapid hardware evolution in the field. Starting from the usage of GPU accelerating techniques [25], [26], and reaching, in some cases like Bitcoin, the development of custom ASICs [16].

However, as we have already pointed out, this has a significant drawback, as the scalability remains the same while the network requires more resources. Ethereum's PoW network's security and vulnerabilities have been highly tested in the past [27]. But the transition to PoS adds more complex logic to the already existing protocols, as this mechanism presents a few critical points that could potentially be exploited in the future.

In the first place, blockchains relying on proof of stake consensus mechanisms highly depend on the network latency [11], and Ethereum is not an exception [28], [29]. Whiling to have hundreds of thousands of messages distributed in a time window of 12 seconds puts a lot of pressure on its peer-to-peer networking layer [30].

Prior research work has tried to consolidate blockchain technology in the Internet of Things (IoT) field [31], [32]. This showed that despite both fields having common native synergies and popular applications such as distributed and trustless ledgers or identity verification. Authors showcase the computation limitations of the IoT devices that could directly compromise the security or the consensus of the network if the resources are not adequately handled, proposing in some occasions to sacrifice in some degree the decentralization in favour of increasing the security [33].

Following the IoT line, Ethereum embraces the transition to PoS, proposing a more complex but less hardware-demanding consensus mechanism, maintaining, meanwhile, its previous security and decentralization. One major target is reducing the hardware requirements for participating in the consensus. The more accessible it is for users to contribute to the protocol, the more resilient it becomes. Previous works have shown how the ranges of routable nodes participating on the Ethereum network oscillate between the 11.000 for the PoW network [34] and 9.000 for the young PoS beacon chain [4]. However, this can't ensure that all those nodes actively contributed to the consensus.

The software industry has always identified diversity as a resilience enhancer method that prevents single points of failure [35], [36]. The diversity concept has been widely used in many industries like the military and the aerospace ones [37], [38], providing resilience even to computer networks [39]. In an effort to make the network as resilient as possible, Ethereum understood that providing software diversity could become a resilience game changer for the network upon future network attacks [40]. Code diversity in the blockchain space has been previously analyzed [41], showing a non-promising conclusion where a significant part of the software is reused for further cryptocurrency projects. However, the study shows that most code reuse generally happens from more mature projects such as Bitcoin or Ethereum to less mature ones. In the Ethereum consensus layer's case, the main specifications are implemented by five main implementations (with an extra non-open-source one), each written in a different programming language.

Finally, previous works have shown the importance and effectiveness of monitoring the hardware resource utilization to identify memory leaks, bugs, and edgy case scenarios that can become a bottleneck for the correct performance [42]. Of course, there is a direct link between the hardware utilization and the overall energy consumption of the machine. Larger hardware needs come at a cost: more energy-hungry hardware. Thus, monitoring these requirements can help understand the overall energy consumption of the software and the network [43]. This is the case of Ethereum, coming from a former PoW scenario, where monitoring the resources of PoS clients can showcase the energy reduction that PoS means to the total carbon fingerprint of the network.

As the network's resilience now relies upon the online and honest behaviour of the validators [44], the software that runs the PoS logic and interacts with the rest of the network becomes a more central piece in the equation. While the execution clients were analyzed in the past [45], in this paper, we study the resource utilization of consensus clients. In particular, we aim to fill the existing lack of a hardware resource analysis of the first transition of consensus mechanism in a live network. The paper presents the evolution of the hardware requirements from the available software to participate in Ethereum's PoS. We showcase the vast reduction of the requirements, deny some myths around the minimal requirements to run an Ethereum node and present the possibility of predicting what is going on in the chain just by checking the allocated resources from the clients.

III. METHODOLOGY

To study differences in the behaviour of the Ethereum CL clients, we have monitored the hardware resource utilization of the six main consensus clients, Teku[3], Nimbus[4], Lighthouse[5], Prysm[6], Lodestar[7], and Grandine[8], in different machines with the most common hardware combinations. Among the studies we have performed, we can distinguish two main phases to monitor the run-time of a client: the synchronization phase and the chain-head following phase. In the first phase, the clients aim to reach the latest state of the chain. Generally, by syncing and verifying each of the blocks in the chain from the Genesis block, clients have to re-compose the chain by asking peers in the network to share the blocks with them. To finally reach the point where they are already up with the head. This process might be more stressful for the machine, as most of the time, the client has to download, process, and verify as many blocks as possible in the shortest possible time. In the second phase, clients are less stressed, as the number of blocks to process happens once every 12 seconds.

By monitoring the hardware resources of consensus clients in different types of machines, this paper aims to track and understand the client's needs, limitations, behaviours, and, thus, performance while participating in the Ethereum network. To do so, the metrics that have been monitored are:

- CPU
- Memory Usage
- Disk Usage
- Network outgoing traffic
- Network incoming traffic
- Syncing time
- Peer connections

A. DIFFERENT USERS, DIFFERENT HARDWARE

To achieve a reasonable comparison of the different clients and, even more importantly, to understand and highlight the minimum requirements for running an Ethereum beacon node, we have selected three different sets of hardware combinations that, from our perspective, summarize the different users that would run a beacon node in Ethereum: *enthusiasts*, *solo stakers*, *staking companies* or *node operators*.

- The first category, *enthusiasts*, are individuals who follow the development of the Ethereum protocol and want to actively support it by running a node in low-power or energy-efficient devices. They don't necessarily need to run a validator but want to find a productive task for their "dusty" *Raspberry Pi*.
- Solo stakers, on the other hand, are passionate individuals who want to contribute to the chain's consensus and run one or a few validators at their own place. They generally have a dedicated personal machine such as an *Intel NUC* or a built PC with the standard components for the personal PC market.

[3]https://github.com/consensys/teku
[4]https://github.com/status-im/nimbus-eth2
[5]https://github.com/sigp/lighthouse
[6]https://github.com/prysmaticlabs/prysm
[7]https://github.com/ChainSafe/lodestar
[8]https://github.com/sifraitech/grandine

- The final target user is the most professional-oriented one, involving users like *staking entities* and *node operators*. These network participants generally participate in the network with a lucrative interest. They generally run hundreds to thousands of validators. Thus, they don't stay short on hardware resources to run them.

TABLE 1. Hardware configuration of the control machines.

Name	CPU	Memory	Storage	Network
Raspberry Pi 4b	4c	8GB	256GB	100Mbit/s
Default node	4c.	15GB	100GB SSD	250Mbit/s
Default node 2	4c.	32GB	900GB SSD	250Mbit/s
Fat node	32c.	120GB	400GB SSD	10.000Mbit/s
Medalla node	1c.	6GB	34GB SSD	100Mbit/s

The hardware resources chosen to test the clients are summarized in table 1. Each of the six main clients ran independently (alone on a dedicated machine) but concurrently on the three available platforms, except for the synchronization of the Medalla testnet, which was done sequentially on a single machine. Of course, the testing times highly depend on the synchronization speed of each client and platform, and each of the three platforms has been monitored on separate dates between 2022 and 2023. Each respective performance phase and platform has its independent subsection in section IV, where the analysis of the results is extended.

B. CLIENT CONFIGURATIONS

In an attempt to increase the network's resilience to human-generated bugs on code, Ethereum has put a considerable emphasis on client diversity. Despite the wide range of client options that users can choose from, finding the right combination of parameters becomes an essential step for node operators to ensure an optimal operation of the nodes. In a complex consensus protocol like Ethereum's, each client can easily exceed the 50 distinct parameters. To reduce the complexity of configuring those parameters by hand without knowing which repercussions they could have on most occasions, clients offer a wide set of pre-defined configurations. Among the most important ones, we can find the following ones:

- **Default:** The default configuration is the base on which each client could work out of the box. It only requires the user to define essential parameters such as the endpoint of the EL, the logging level, the destination of the database, whether the user wants to save the logs on a file or if the user wants to serve debugging metrics on a specific port (for internal monitoring dashboards).
- **All topics:** Using the base configuration as a reference, most clients provide the possibility to subscribe to all the attestation subnet topics. This ensures an optimal network presence of the node, ensuring that the attestation of a validator can easily propagate over the network. This mode is generally targeted for node operators with multiple sets of validators. However, it has a few drawbacks: i) as more messages arrive from an increased number of GossipSub [46] topics there is a noticeable increase in the CPU usage for validating those messages, and ii) as the number of messages that the node has to receive and then propagate back to the network increase, so does the bandwidth usage of the node (both in and out network traffic).
- **Archival:** The primary objective of the beacon chain is to finalize *beacon states* as the chain grows, meaning that the status of all validators is immutable as the chain keeps evolving. The consensus contemplates the possibility of not finalizing, as the network could be divided across forks and eventually would have to converge on one of them as canonical. In such scenarios, clients save beacon states on disk as checkpoints in case they need to roll back or recompose anything in the past. In their default configuration, clients have a frequency at which these checkpoints are stored locally. This frequency, which tends to be around the 8192 slots by default, can be adjusted at the client's parameter. Reducing this parameter between 1 and 4 epochs (32 to 128 slots) is considered spawning an archival node. This mode considerably improves the response time of queries through the REST API, and it is generally the preferred mode for researchers or chain indexers despite the larger storage usage.

Client developers ensure that their client works optimally on the default configuration, i.e., setting a proper target of peer connections, a proper beacon state check-pointing frequency, or an optimal cache sized, even the simplest database schema that could fulfil most of the use cases. Thus, for most users, it is recommended to stay within the margins of these default parameters, changing only those ones they know they need for their particular needs.

C. WHY ETHEREUM HAS SO MANY NETWORKS?

Complex protocols such as Ethereum are not easy to consolidate in the technical or the human aspects; they have too many parts and contributors. Thus, organising any possible upgrade for the protocols is even more complex, as the community must coordinate which changes have to be done, who reviews them, and how to test them. Like many other projects in the web3 ecosystem, Ethereum adopted a basic methodology of splitting all the ideas into features or Ethereum Improvement Proposals[9] (EIPs). When a group of proposals or EIPs are considered mature enough and align with the public roadmap of the protocol, they are tested on development environments such as testing networks or, as we refer to them, testnets.

The large trajectory of Ethereum means that there have been many EIPs and, therefore, testnets. This testing environment represents an isolated environment where a few specific changes, events or transitions can be stress-tested. Although the testnet aims to represent the reality of a main network as closely as possible, some aspects or events, like a

[9]https://eips.ethereum.org/

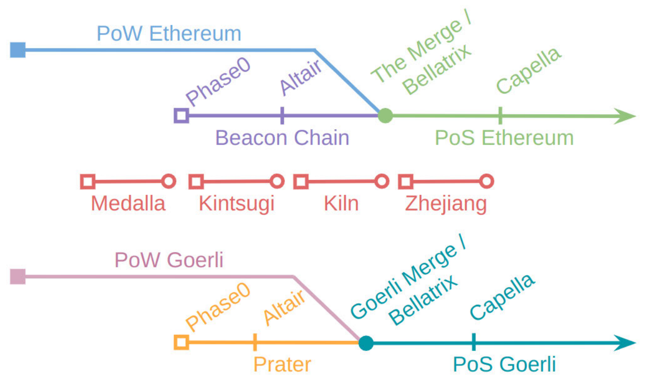

FIGURE 4. Ethereum network's hard-forks organization, public testnets in red.

sudden peak in user interaction, can't be entirely replicated in these environments. However, because these testnets justify their presence by testing the limitations of the changes, it doesn't mean that their results are negligible. In fact, they are generally stress-tested to find the weak points of the protocol or the implementation. Figure 4 shows the road map of the main public testnets around the consensus layer's hard-forks timeline, showcasing how each major upgrade on Ethereum was previously tested on a testnet. It is important to notice that in public testnets such as Goerli,[10] participation is not monetarily incentivised. Thus, participation tends to be lower than on the main networks. This might not sound that critical if it stays above the 66%. However, these networks became the best method to test "unexpected" scenarios like forks and non-finality periods originating from code bugs or weird interactions. As these edgy cases generally exercise the hardware of the nodes in unusual patterns, they are relevant to visualize which are the maximum requirements.

D. EXPERIMENTS TIMING SETUPS

This paper presents a large number of experiments, including multiple client configurations, operational stages, and different stages or hardforks of the chain. The following paragraphs summarize all the information about when we ran these experiments, for how long, and which hardware where they were running on.

1) SYNCHRONIZATION ON DEFAULT AND FAT NODES

The chain synchronisation from genesis was performed using the available six clients on *default mode* III-B in a "default node" and "fat-node" from Table 1. The data was collected between the 8th and the 18th of March, 2022, with the following client versions:

- Prysm: 2.0.6
- Lighthouse: 2.1.4
- Teku: 22.3.2
- Nimbus: 1.6.0
- Lodestar: 0.34.0
- Grandine: 0.2.0

[10]https://goerli.net/

2) REGULAR PERFORMANCE ON A REGULAR NODE

The experiment included running an Execution Client and the Consensus Client on the same machine. To promote a fair comparison between the CL clients, we paired all of them with a Nethermind node. The study was performed between the epochs 201699 and 202699, or in a human-readable format, between the 17th of May 2023 and the 21st of May 2023. The experiment was conducted in the Goerli network, running each of the clients' pairs (listed below) on a "default node 2" from Table 1, using the client versions listed in the following list.

- Prysm: 4.0.3
- Lighthouse: 4.1.0
- Teku: 23.4.0
- Nimbus: 23.5.0
- Lodestar: 1.8.0

3) SYNCHRONIZATION OF THE MEDALLA NETWORK

The synchronization of clients in the Medalla testnet was performed between the dates displayed in Table 2 (prior even to the official launch of the Beacon Chain). The clients were run sequentially in a "Medalla node" from Table 1 using the following list of versions:

- Teku: v0.12.14-dev-6883451c
- Prysm: v1.0.0-beta.1-4bc7cb6959a1ea5b
- Lighthouse: v0.3.0-95c96ac5
- Nimbus: 0.5.0-9255945f
- Lodestar: commit 40a561483119c14751

TABLE 2. Running dates of each client during the synchronization of the Medalla network.

Client	Start Time	End Time
Teku	2020-11-09 17:25:12	2020-11-10 17:34:45
Prysm	2020-11-04 18:34:12	2020-11-06 09:34:34
Lighthouse	2020-11-02 17:17:51	2020-11-04 02:57:38
Nimbus	2020-11-04 18:40:35	2020-11-06 10:23:04
Lodestar	2020-11-08 20:19:02	2020-11-09 08:54:04

E. TOOLING

This paper relied on sophisticated and well-tested tools to generate and measure the data presented in evaluating the CL clients with precision and reliability.

1) NODE EXPORTER

Each control machine we have used to perform the study has been entirely monitored by the software tool *Node Exporter* [47], giving us access to the whole list of metrics mentioned in the previous paragraphs. Combined with a central *Prometheus* [48] time-based database service that scrapes each of the exported endpoints every 15 seconds, it can accurately provide the resources in use for an entire machine.

As calibration for the software tool, we have benchmarked the results obtained from the node exporter with the ones taken from a *Python script* that reads the same metrics in run-time from the machine every second. The successful comparison between the values of each gathering tool demonstrated that the node exporter showed a negligible overhead to the measurements while providing the confidence of using a reliable, unbiased, and well-tested tool to monitor the resources of each machine.

2) API WORKLOADER

It is essential to mention that running an Ethereum node doesn't only concern users or entities that want to run validators on top of them. Accessing data from the blockchain is as important as reaching a consensus on the chain. Thus, many companies and researchers are actively participating in the network with the final goal of accessing, analyzing, or simply selling chain-related data.

Most users access on-chain data such as transaction status, chain status, and validator performance through chain explorers such as Etherscan,[11] EthSeer[12] or Beaconcha.in.[13] However, to support these web applications, they must interact with chain data through the REST API that most clients offer. There is a defined standard number of endpoints [49], [50] that the clients are supposed to offer. However, the performance and reliability of retrieving that information from the API are not standard among the clients.

Since quick access to this data might be essential for some users, as we define the performance of an API, we developed an API Benchmarker tool [51] that assesses any given REST API endpoint's responsiveness and robustness under varying conditions and workloads. Intending to make the fairest comparison across clients, we benchmarked the response time of each client's APIs using their archival mode (only for those who had it). The benchmark was implemented by performing multiple queries at the same endpoint for different clients. The selected query was $/eth/v1/beacon/states/[slot]/validator\_balances?id = [validator\_id]$, which forces to recalculate a beacon state in the past, returning the balance of a given validator. All the clients were asked the same queries in the same order, which were selected at random to ensure the recalculation of the past beacon state.

IV. HARDWARE RESOURCES' EVALUATION

To extend into a larger and more detailed analysis of how the Ethereum PoS transition has impacted the hardware requirements to participate in the network and consensus, we have accumulated almost 30,000 CPU hours (3.4 CPU years) across Ethereum's CL clients in different hardware and configurations. The study includes data from over a year, where we have collected over 735 million data points, from which we extracted almost 150 million data points to plot about a thousand different figures showing how the other CL clients perform.

[11]https://etherscan.io/
[12]https://ethseer.io
[13]https://beaconcha.in/

The following section discusses our findings, reviewing the most critical points that we identified in our journey into Ethereum's PoS transition. We have summarized or compressed the graphs and plots to the minimum for time and space reasons. However, following the transparency and open-source standards of the Ethereum community, all the figures are accessible on the following GitHub repository [52].

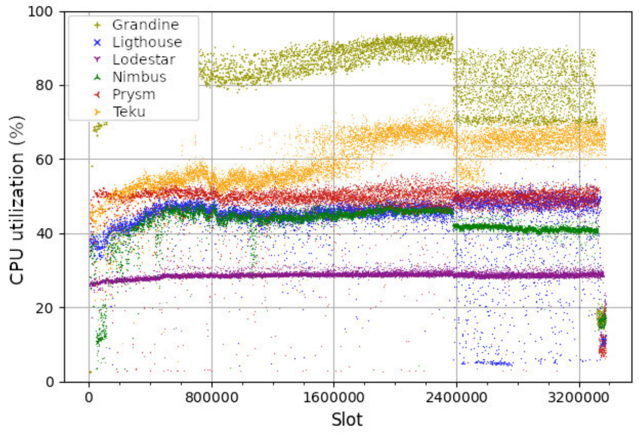

FIGURE 5. **CPU utilization while syncing the chain from a default node. Comparison across clients.**

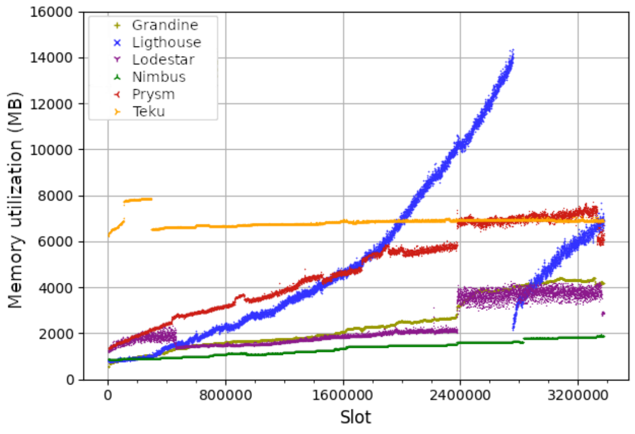

FIGURE 6. **Memory utilization while syncing the chain from a default node. Comparison across clients.**

A. SYNCHRONIZATION PROCESS

The entire chain synchronisation is the event of downloading and verifying the historical blockchain from the genesis block to the head of the same one. It is a process that only gets done at the beginning of the client's lifetime and that could be repeated on rare occasions, such as when this client is changed to a different one. Nevertheless, it is still essential for the network to ensure the existence of full nodes in the network with all the historical data downloaded. Thus, the community needs to monitor the chain synchronization from Genesis, as it can help users better estimate the preparation time of each node before they can host any validator.

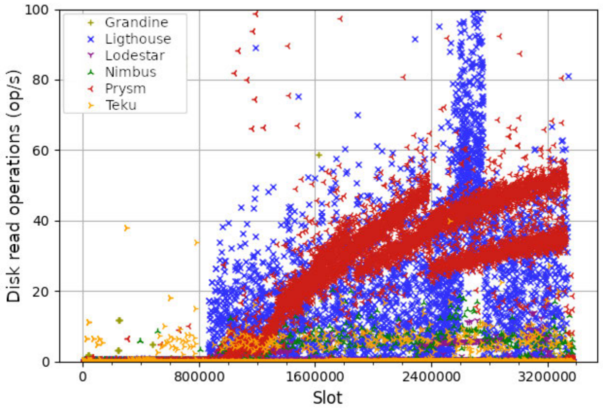

FIGURE 7. Disk read operations/s while syncing the chain from a default node.

Full nodes ensure the blockchain is composable and verifiable at any given time. Therefore, it is essential to ensure that the process is not slow, long, and tedious if users want to sync the entire chain from scratch. Mainly when the downtime of an already activated validator might depend on the downtime of the Beacon Node. To measure the different synchronization techniques proposed by the community, we have monitored this process for the leading clients in the Ethereum CL ecosystem.

The following evaluation paragraphs refer to the gathered under the details described in Section III-D1. It is essential to mention that by the time we made this measurement, it wasn't necessary to pair each CL client with an EL client to keep the head of the chain correctly. Thus, all the metrics displayed in this section will refer only and exclusively to the resources used by the CL clients.

1) CPU UTILIZATION

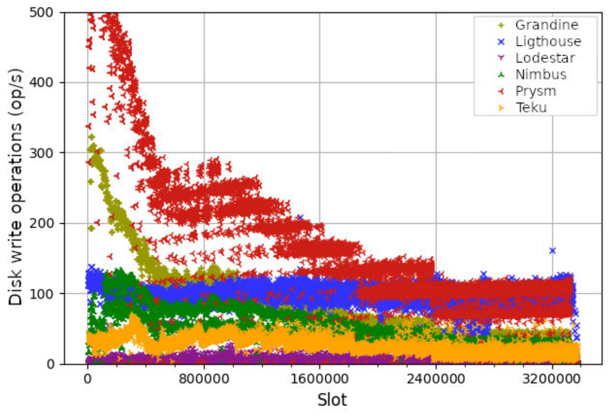

FIGURE 8. Disk write operations/s while syncing the chain from a default node.

The implementations of Ethereum's CL specifications are written in different programming languages, which can ultimately determine critical parameters such as the level of concurrency the program can achieve or the optimization level for the validation and underlying processes. Figure 5 shows the CPU utilization degree achieved by the CL clients using the chain slots as a reference; this helps to have a fair comparison between clients, as some finished sooner than others. In the figure, we can appreciate different CPU profiles, where Grandine follows a different recognizable pattern, using around the 80% of the CPU while syncing. Teku follows the lead on CPU utilization with a slightly increasing ratio reaching the 60% of the CPU. The figure shows similar CPU profiles for Prysm, Lighthouse, and Nimbus, leaving Lodestar as the client that requires less CPU to synchronize the chain.

There is a crucial point to highlight here; although we could associate a high CPU utilization with a drawback to a CL client (we come from a premise where reaching consensus in PoS requires very few resources), it isn't necessarily bad. Not at least if the CPU cycles are optimized to sync the chain faster. Remember that the sooner we can sync the chain from Genesis, the lower the downtime users could experience on their validators. Thus, we support the idea of squashing the available resources during the synchronization phase if it reduces the duration of the same one.

2) MEMORY UTILIZATION

In terms of memory, all the clients presented different behaviours. Figure 6 shows the memory allocation patterns from the CL clients. The figure shows that most of them allocate more memory throughout the synchronization process. This is an expected pattern, as the beacon state[14] grows as time passes, and more validators tend to join the chain. Among all the recorded patterns, it is clear that Lighthouse has an unusual, constantly increasing one. Reaching even the limits of the machine, what we can identify as a possible memory leak, leading to a crash and a forced restart of the machine. Of course, the incident was shared and reported with the Sigma Prime team[15] (developer team of Lighthouse), and it is known to be fixed in the following version $v2.2.0$.

On the other hand, it wasn't the only client that reported some difficulties when setting it up. Teku and its associated JVM are somehow tedious to configure. The JVM requires a minimum amount of memory to work and experiences sudden crashes if not enough memory is provided. However, we could achieve a steady performance by assigning $6GB$ of memory to the JVM. As represented in the figure, Teku maintains reasonably constant memory utilization, never exceeding $7GB$ of memory. The memory profiles get more stable with the rest of the clients, where only Prysm reaches the same level as Teku, and Lodestar, Grandine, and Nimbus keep a lower profile at a lower limit of $4GB$. It is worth mentioning that Prysm is written in *Go*, which tends to allocate and keep the memory for the processes until the OS asks it back. Thus, this makes the comparison a bit unfair for

[14]The beacon state represents the state of the chain at a given slot. It includes the status, balance, and information of each validator.

[15]https://sigmaprime.io/

Prysm, as part of the measurement might belong to unused but still allocated memory by *Go*. In these lower memory profiled clients, Nimbus has the lowest memory consumption, with a steadily increasing profile that keeps under 2*GB* of memory allocation and shows an incredible memory optimization for such an intense process.

Similarly to the CPU utilization profile, we can identify where most clients start to behave differently. Around slot number 2.4 million, this point refers to the transition to Altair's hard fork in the Beacon Chain. From Altair, the CL clients need to track sync committees and other significant changes in slashing conditions, which explains the difference in resource consumption. The overall block size also increased, making it heavier to download and slower to process while keeping more bytes in memory.

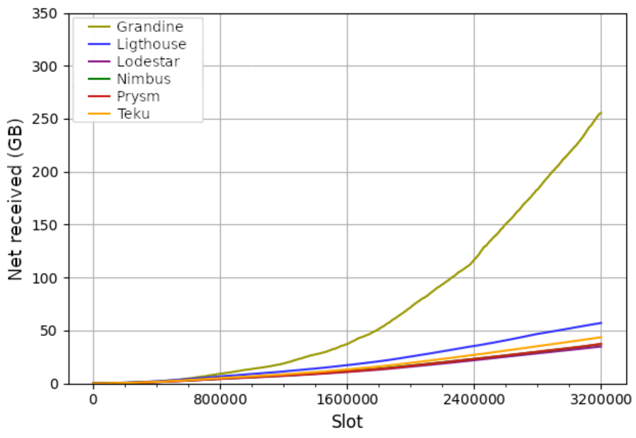

FIGURE 9. **Network inwards utilization while syncing the chain from a default node.**

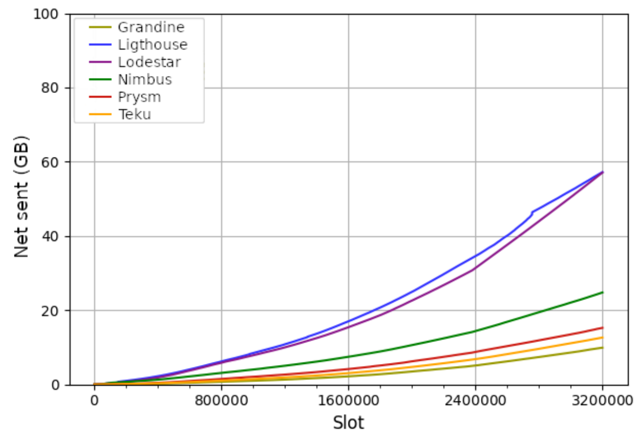

FIGURE 10. **Network outwards utilization while syncing the chain from a default node.**

3) DISK UTILIZATION

The differences are pretty impressive when comparing the distinct size it took to keep the entire chain by the clients. Table 3 shows the average disk usage using the slots as a reference. Although all the clients have run on their default configuration, the figure shows that there is an important difference in how much storage the different clients require; for instance, Lighthouse takes over three times the storage of Teku, which is the CL client that requires the least storage followed by Nimbus. There is a catch: although all the clients are in charge of keeping the whole set of raw blocks on disk (raw blocks databases barely change in size across clients), there are differences in the frequency of maintaining beacon state checkpoints. We have already mentioned that the beacon state represents the chain snapshot at a given slot. It changes over time as more blocks are added to the chain. Thus, there is no need to keep it in disk with a super low frequency, as its size generally is much larger than a single block. For this reason, beacon nodes keep track of beacon state checkpoints, and when they need to access a specific state from the past, they can load the closest state they have stored, applying the subsequent list of blocks until they can regenerate the desired state.

All this said, we can deduce from Table 3 that the default checkpointing in Lighthouse has a higher frequency, keeping in disk more checkpoints for the same range of slots, which ultimately helps access faster any state in the past. It is important to note here that this parameter is adjustable for all clients, and it might be important to tune it up based on the necessities and resources of each user.

TABLE 3. **Total disk usage of the beacon node's database keeping beacon blocks and states after syncing the chain on the default mode.**

Grandine	Lighthouse	Lodestar	Nimbus	Prysm	Teku
64GB	105GB	81GB	46GB	62GB	34GB

The disk utilization has some other peculiarities, though. With a wider focus on disk write operations in Figure 8, most clients, and in particular Prysm, have a higher number of disk write operations per second at the beginning of the syncing process, and this decreases gradually until it plateaus after Altair. This is explained by the fact that at the beginning of the Beacon chain, the number of validators (21.063) was significantly lower than 3.2 million slots later (295.972), making the whole slot processing of the blocks much faster. Both figures 8 and 7 are inversely correlated; the increase in disk reads is relatively low in the first half and accelerates in the second half, as the number of validators and attestations in the network increases. The measurements in Figure 7 show that disk read operations increase dramatically for most CL clients, except Teku. This is not an issue because it does not affect the performance. But it happens virtually simultaneously for most clients around slot 900*K*, being Prysm client to more clearly expose this behaviour.

We attribute this pattern to some falling caching techniques that might force the client to read the needed information from the internal client cache, producing more read operations.

4) NETWORK BANDWIDTH

Network connectivity in a distributed network is a parameter highly associated with the number of concurrent connections the node keeps with others in the network. Despite being in the synchronization phase, Ethereum is not different from other networks in that aspect. Figures 9 and 10 show the aggregated received and sent *GB* throughout the synchronization of the chain.

Figure 9 shows that most of the clients follow a similar ratio of total downloaded GBs to sync the chain around the total of 50GB. Grandine, on the other hand, outstands its competitors, requiring five times more bandwidth to perform the same task. Handling more concurrent downloads to sync the chain faster seems to be a not-that-optimized deduplication of downloading multiple times the same blocks.

Regarding sent GBs, Figure 10 shows that only Lodestar and Lighthouse are above the rest of the clients with around five times more output bandwidth than the ‘‘quieter’’ clients.

TABLE 4. Target of peer connections for each client during the chain synchronization on a default configuration.

Lighthouse	Lodestar	Nimbus	Prysm	Teku
45	25	155-160	45-55	75

As mentioned, bandwidth usage highly depends on the number of simultaneous peers each client has. Table 4 shows the target of peers each client got during the syncing process. The table shows that by default, Nimbus looks for stable 150 peers from where to fetch blocks. Followed by Teku with 75, Prysm with 50, and Lighthouse and Grandine, which share a peer target of 45. To finish, Lodestar has shown the lowest ratio of clients with a stable target of 25 concurrent peers.

5) PERFORMANCE

We have presented many insights about how each client operated over this first step of connecting to Ethereum's network and synchronizing the chain. We have seen many different design choices to perform this same task, like choosing to keep more data in the cache to reduce the number of interactions with the disk (the slowest task of a computer), increasing the number of concurrent processing tasks to minimize the total time, or optimizing the process to make it as light and fast as possible. Under our interpretation, we define as ‘‘performance’’ the ability of a client to sync the chain in the shortest time possible without capping the hardware resources of the matching running it.

Figure 11 shows each client's total duration of the synchronization process. The figure shows that concurrency clearly benefits the process. Grandine was the fastest client to catch up with the head of the chain in almost 50 hours. The fastest clients are closely followed by Nimbus and Prysm, with around 90 hours, displaying that it is possible to achieve great timings if optimization, concurrent downloads from multiple clients, and a mild level of CPU usage are achievable. Lighthouse and Teku have also achieved similar results with around 135 to 150 hours-long process. It is not a fair comparison for Lighthouse, as it crushed for an over-allocation of memory and had to be restarted. However, the figure clearly shows that it might be the most trustless client of the set, investing in keeping more states on disk more often in case a chain reorganization happens. Teku and Lodestar are the least optimized clients catching up with the chain's head. Despite the highest memory allocation, the second highest CPU utilization, and the second highest number of peers from Teku, it still got in fifth place. In the last place, the measurements show that Lodestar has one of the lowest profiles of the clients, perhaps aiming for a light client that could run in the background.

B. THE BIGGER, THE BETTER?

As anyone could expect, having a more capable machine has its benefits. The majority of the tested clients take advantage of parallelization techniques to speed up the entire synchronization process. However, some implement concurrency techniques better than others, taking a huge advantage as they can sync up the historical chain sooner. With that many parameters involved in the performance of a client, increasing each of them individually can vary the benefits differently:

- A faster CPU with more cores allows the client to process attestations at a higher rate and, therefore, more blocks simultaneously. However, there is a point that having many CPUs is not useful since many cores may stay idle if there aren't that many events to process and validate.
- A bigger memory available while syncing increases the number of items the client can keep without performing slow read and write operations to disk.
- A higher network bandwidth can allow us to keep more simultaneous connections with peers in the network, enabling the concurrent download of more blocks from the chain.
- A faster disk can reduce the bottleneck originated by writing such a long chain as the Beacon Chain.

1) SYNC SPEED

Our measurements show that machines with superior hardware resources can synchronize the chain faster. However, with the apparition of checkpoint syncing, the whole operation of syncing the chain from Genesis can be reduced to almost zero times. Some CL clients offer a back-filling sub-process to fetch the historical chain backwards from the given checkpoint. However, this is not the case for all of them, and if users want to have faster access to on-chain data, known as ‘‘archival node’’, they might still be forced to perform a full synchronization from Genesis. This ‘‘archival’’ mode increases the frequency of the chain checkpoints being stored in the database, making all the RPC queries generally faster as it takes less time to recompose intermediary states. It is ideal for users who want faster access to chain data. The whole concept of checkpoint syncing relies on the node's

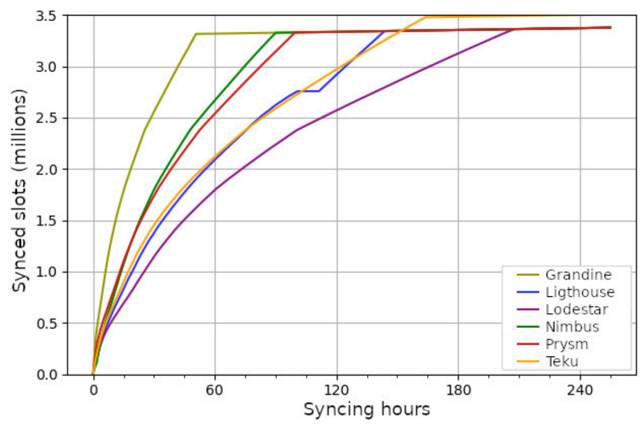

FIGURE 11. Chain synchronization speed as the number of slots downloaded and processed since the node on standard hardware was run.

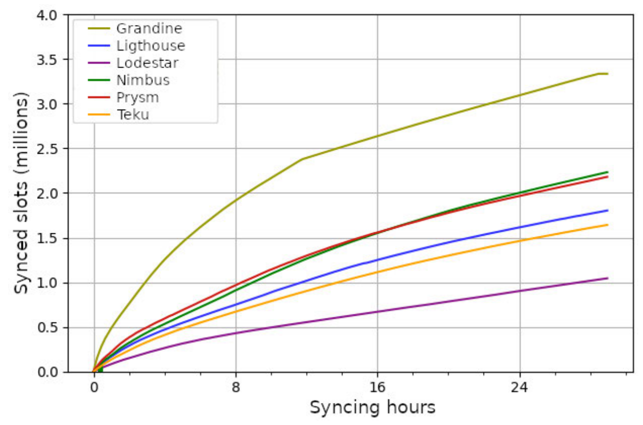

FIGURE 12. Chain synchronization speed as number of slots downloaded and processed since the fat node was run.

single need to access the last beacon's finalized state to operate. This process allows a node to fetch the last finalized Beacon State and Signed Beacon Block from a trusted node, allowing it to process and validate incoming new blocks after the process has been accomplished.[16] Nevertheless, although checkpoint-syncing is the recommended operation to catch up with the chain's head, not all users can access an already synced beacon node from where to fetch the last finalized checkpoint, making the full synchronization speed still important.

Figure 12 shows that clients achieve a better synchronization speed with more capable hardware. On average, our measurements show a speed improvement of the 135%, with Lodestar showing a disappointing performance decrease of a 10% and Grandine syncing a remarkable 178% faster. All the measured slot processing ratios per second are displayed in Figure 13, where we can appreciate the average performance of Teku, Nimbus, Prysm, and Lighthouse, and the outlying one from Grandine and Lodestar.

2) DISK UTILIZATION

Of course, having more and faster processing power implies more disk reading and writing operations (if no disk bottleneck is hit). In the previous subsection IV-D, we appreciated how, around slot 900.000, the disk reads spike until the end of the process. Controlling the same metric on the fat nodes, Figure 14 shows almost the same behaviour, but this time, the pattern started much later, around slot 2.000.000. The fact that the phenomenon occurs later on in nodes with more memory makes us believe that some memory caching process is originating or preventing these disk read operations. For this reason, when the client reaches its buffering limit, it starts generating a significant amount of disk reads.

On the other hand, disk write operations remain on the same pattern as Figure 33 represents, increasing its offset with the faster synchronization speed. This showcases that although we need a proper or fast enough disk to handle an Ethereum CL node, unless we want to set up an archival node, there is not much impact on disk usage that originates from increasing the rest of the hardware components.

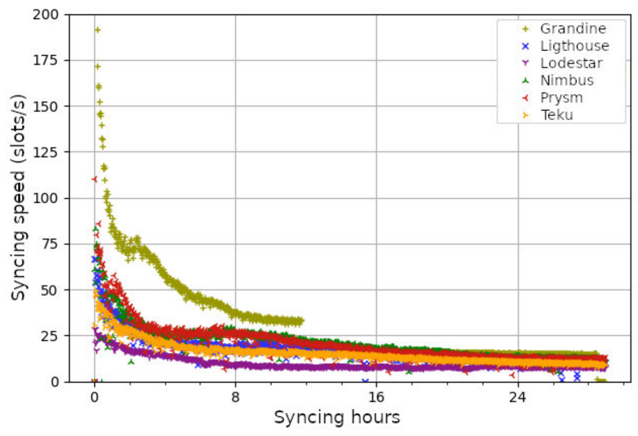

FIGURE 13. Slot synchronization ratio per second for the different clients.

C. HOW SMALL COULD WE GO?

Despite the complexity increase that shifting from PoW to PoS means, the protocol has bragged about requiring less computational power to reach consensus than its PoW predecessor version. Going even to the limit of stating that a Beacon client could run on a Raspberry Pi. We have tried syncing up the mainnet chain from Genesis using the different clients, with the unanimous conclusion that it is an impossible task to achieve, based on the large time it takes to finish it (exceeding the 15 days to sync on the fastest client). The beacon chain at that point, March 2022, had around 3.400.000 slots, which presented a major problem to the restricted hardware of the Pi. The chain at that time had around 550.000 validators, making the last steps of

[16] Although the beacon node or the CL client can process new incoming blocks after a checkpoint sync, its performance is still subjected to the synchronization of the paired EL client. EL clients, when writing this paper, can not sync from a checkpoint state. Thus, the time needed to have an operative EL + CL client relies on how fast the EL can catch up with the head of the chain.

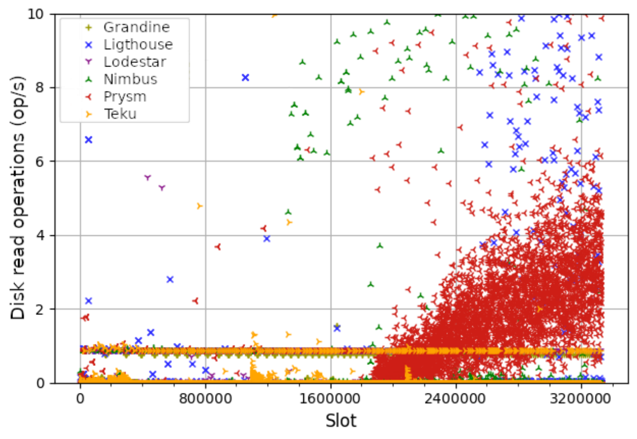

FIGURE 14. Disk read operations by the clients while running in a "Fat" node.

the synchronization tedious because of the state processing duration. For timing reasons, we've extended the paper to explore the performance of the synchronization process in the Kiln[17] testnet in the Appendix.

Taking into account that the Beacon State is append-only, we expect that the required hardware resources are just going to increase over time. As a result, CPU and memory requirements will increase with bigger state transitions to process. Even if the number of validators decreases, which doesn't seem feasible, at least in the short term, new updates to the protocol will always require more validations or tasks. Following this trend, after the merge, each consensus client node needs to be paired up with an execution client to follow up the head of the chain successfully. This ultimately means that the available resources have to be duplicated to run both parts of the system together.

Although there is a tendency to require faster and more extensive resources, we are definitely seeing a massive reduction when comparing it to the predecessor PoW consensus version. It is not crazy to think about having a *Raspberry Pi* set up to participate in the network. However, having to purchase a faster SSD and probably extending to a secondary *Raspberry Pi* to run the execution layer client (with its dedicated faster SSD) makes it, from our point of view, unattractive for most of the users. In this case, we suggest opting for a still low-profiled and low-powered device such as an *Intel NUC* or similar.

D. REGULAR PERFORMANCE

Having the possibility to fully sync the chain from scratch and ensure that doing so is viable is a nice feature of the community. However, as we have introduced previously, current node deployments can benefit from syncing the chain using chain checkpoints.

At any point, the resources taken from a client at a regular workload (following the head of the chain) are considerably smaller than those they need to sync up the chain. This subsection describes the resources that each client needs to perform after the synchronization phase.

To study a fairly more accurate representation of the actual consumption of the set of clients, we have also measured the resources each of them needs to participate in the network while following the head of the chain. Further details on the study dates and the setup configuration are defined in Section III-D2.

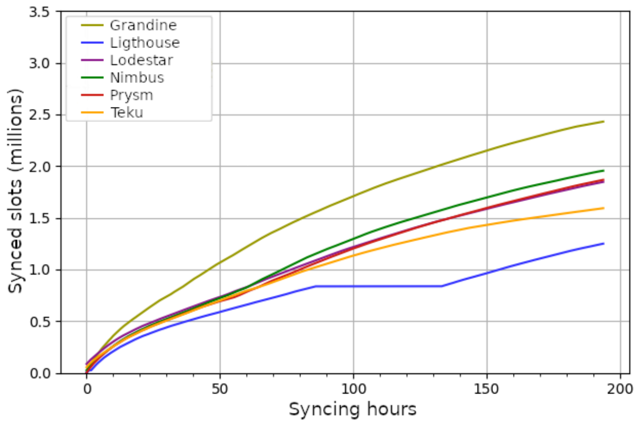

FIGURE 15. Chain synchronization speed as the number of slots downloaded and processed since the node on the Raspberry Pi 4 was run.

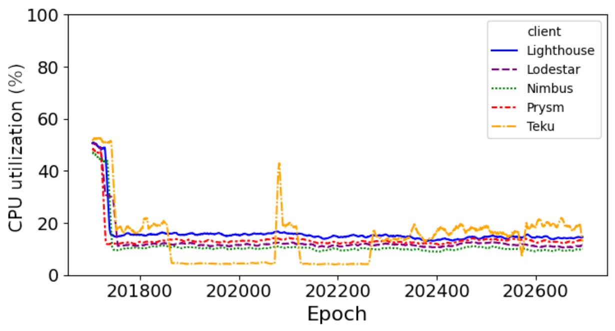

FIGURE 16. CPU usage by clients following the head of the chain.

1) CPU UTILIZATION

Unlike the chain synchronization process, the regular operation of EL and CL clients doesn't require much CPU. Figure 16 shows CPU usage by each of the combined clients as the percentage from the total available in the machine. The first slots in the figure show that despite syncing from a trusted node's checkpoint, validating it and taking it up with the head of the chain is way more demanding than just following it up. The next epochs in the chart show that both CL and EL need less than 40% of an eight-core machine to handle the propagation of blocks, attestations, and aggregations, as well as the validation of the same ones and recomputing the state. Bast reduction in contrast to its previous PoW consensus mechanics before the merge.

[17]https://github.com/eth-clients/merge-testnets/tree/main/kiln

There are a few more insights to take from the figure, though. Despite the difference being not much across the CL clients, some are more efficient than others. Nimbus, Prysm, and Lodestar are the most efficient clients, taking between 11% to 13% of the CPU to perform all the consensus tasks. On the other hand, Teku and Lighthouse are the ones requiring more CPU, with Lighthouse keeping an 18% of average CPU usage and Teku being the most unstable client, fluctuating its CPU usage between 15% and 23%, sometimes reaching 30% of use.

We have to mention, though, that there were some infrastructure problems with Teku, which crashed two times during the experiment due to a lack of space for the machine. These interruptions are clearly visible in the figure, wherewith sudden drops in Teku's CPU usage to a flat and stable 3% to 7%, which we can attribute to Nethermind. Spikes of 40% to 60% also follow these drops, which are related to the client catching up on the blocked missed while it was down.

2) MEMORY UTILIZATION

More different memory usage patterns were measured on the machines during the study. In contrast to CPU usage, memory usage increases once the client follows the chain's head. Figure 17 shows that the client difference is more noticeable than the CPU one. Nimbus and Lighthouse are the most optimized clients with averages of (remember this is the aggregation of memory between Nethermind and the clients) 15GB and 18GB, respectively. Lodestar follows them quite closely, with an average of 19GB. Leaving Prysm and Teku as the highest memory-dependent clients with 20GB and 23GB of memory usage, respectively.

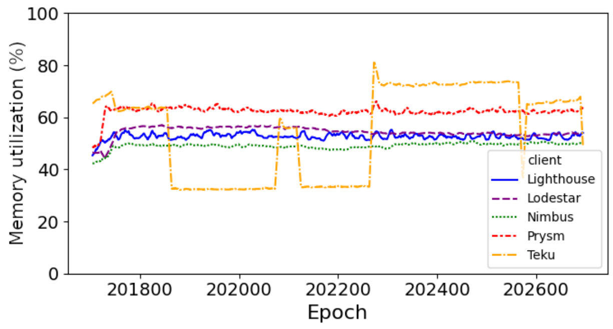

FIGURE 17. Memory usage by clients following the head of the chain.

Please note here that Teku is the only client that requires less memory to follow up the chain than to sync it up, and that part of the distribution shows the flat memory usage of Nethermind of 11GBs.

These more updated measurements show that Lighthouse has fixed the memory leak and that even though the CPU is heavily used to sync up the chain, a bigger cache or more memory is used more by clients when performing under a normal state of the chain while following it.

3) DISK UTILIZATION

From the perspective of the interaction with the disk, there are significant differences between clients. In contrast to the disk usage displayed in Section IV-A, where clients could handle the persisting of blocks and states to DB in a more customized or optimized way, neither Figure 18 nor Figure 19 show clear differences in the number of Bytes read or written into the disk. Figure 18 shows a general 2MB per minute ratio of readings from the disk with some sporadic spikes from 4MB to 10 MB per minute. On the other hand, Figure 19 shows that writings are below 1MBs per minute, with more often spikes from 2MBs to 10MBs per minute. This clearly shows that despite keeping more items and data in memory, the operation of a client requires way more reads and validations than actual writings to disk.

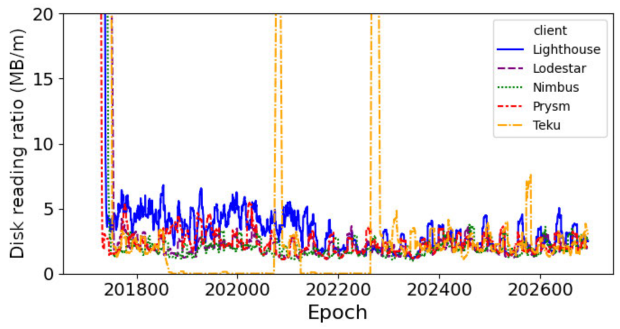

FIGURE 18. Disk writes by clients following the head of the chain.

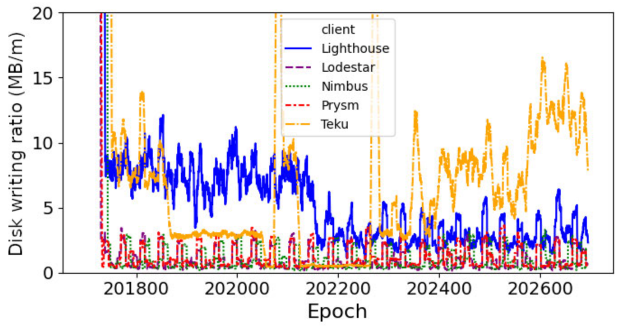

FIGURE 19. Disk reads by clients following the head of the chain.

4) NETWORK BANDWIDTH

Previously, We've seen that the resources aren't anything out of the normal computer standard, which means that the combo of EL and CL clients could efficiently run on modern pre-manufactured PCs or computers. However, can a regular router handle all the communication involved?

The number of sent and received bytes over the network while following up the chain is directly proportional to the number of connections opened with other nodes in the network. The more nodes you connect to, the more messages or requests you receive and share. It doesn't matter if they are duplicated messages during the propagation of blocks or attestations; nodes will see an increase in network usage.

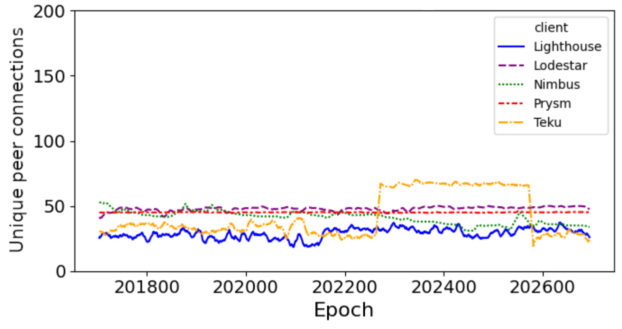

FIGURE 20. **Concurrent node connections by the clients while following the head of the chain.**

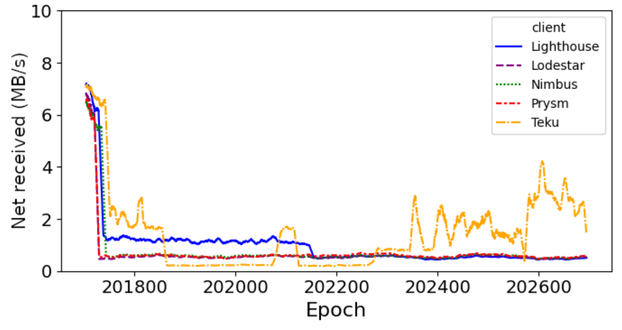

FIGURE 21. **Network incoming bandwidth for the clients while following the head of the chain.**

Table 5 shows each client's default target amount of peers. The measurements show that most of them, including Nimbus, Prysm, and Lodestar, have a similar target of around 50 peers, which generally keep steady except for Nimbus, which sees a small drop of connected peers at the final epochs of the study. On the other hand, Teku and Lighthouse fluctuate more between 30 and 40 concurrent connections. Comparing these values to the previous ones presented in Table 4, there is a noticeable decrease in the number of peer connections. The most representative case is that of Nimbus, which went from 150 connections at the beginning of its journey to 50 nowadays. We interpret these adjustments as optimising a "too ambitious" prior target that doesn't justify the extra bandwidth cost.

TABLE 5. **Target of peer connections for each client during its regular operation.**

Lighthouse	Lodestar	Nimbus	Prysm	Teku
30-40	50	50	50	30-40

To observe the impact of these concurrent connections in the downlink, we can use Figure 21, which shows the amount of received MBs per minute for each client combo. Leaving the first synchronization from the checkpoint range, the figure shows that most clients, Nimbus, Prysm, and Lodestar, remain constant under one MB per minute. Clear contradiction to clients with more unstable peers (in terms of peer connections) like Teku and Lighthouse, which stay above the 1 MB mark, touching the received 2MBs per minute. This may look like a contradiction since fewer peers send more traffic. We attribute this phenomenon to the fact that establishing a connection, which involves making the handshake and sharing certificates to ensure encryption between both partners, can be more demanding than just sharing messages. In this case, Teku is the chattiest client.

Figure 22 shows the impact in the uplink by providing the sent MBs per minute by the CL plus EL combo. The figure shows how most clients share a similar pattern, with Prysm being the chattiest client.

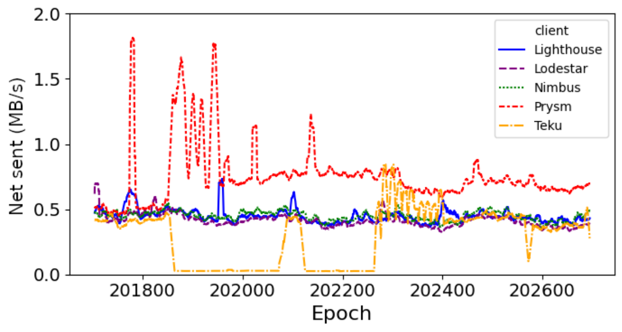

FIGURE 22. **Network incoming bandwidth for the clients while following the head of the chain.**

V. HARDWARE AND NETWORK OBSERVATIONS

The wide variety of data presented in the previous section IV compile the raw resource utilization prints that different clients infer in different machine configurations. These studies provide a wide knowledge of the minimum requirements and the optimal hardware needed to participate in Ethereum's network. However, many other aspects can be directly linked to a specific resource utilization pattern. As presented in the following subsections of the paper, we analyse how the Ethereum protocol, network instabilities, or the user's interaction impact the resource utilization of the different clients.

A. CPU AT SLOT TIME UTILIZATION

We have previously introduced the overall resources needed by each client while syncing and following the head of the chain. However, we haven't mentioned which processes trigger such intense use of CPU during the performance of a client. As mentioned, Ethereum's PoS defines a new chain organized in slots and epochs, where active validators split their duties across the epoch. This means that in every slot, a subset of active validators is in charge of receiving and validating the proposed block, having to submit the attestation votes then, and the aggregated attestations. Despite the new PoS' Gasper fork choice [53] defines the propagation and arrival of these duties as asynchronous, the rewards clearly incentive validators [44] to congest them in defined periods (not mandatory, but ideal for maximizing the rewards for each of them). A detailed description of the expected time windows for each operation is described in Figure 23. In the figure, we can appreciate how each duty is followed by a

4-second window to propagate each message over the network.

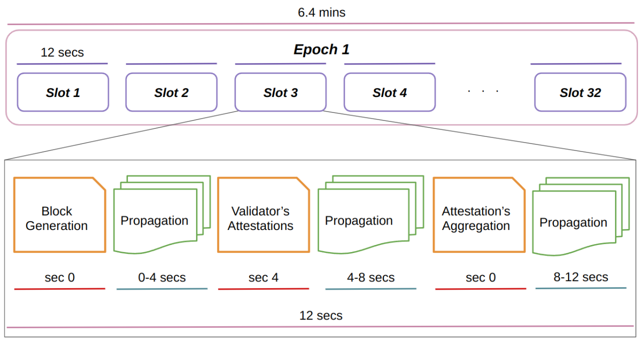

FIGURE 23. Description of the different validator task's time windows inside a single slot.

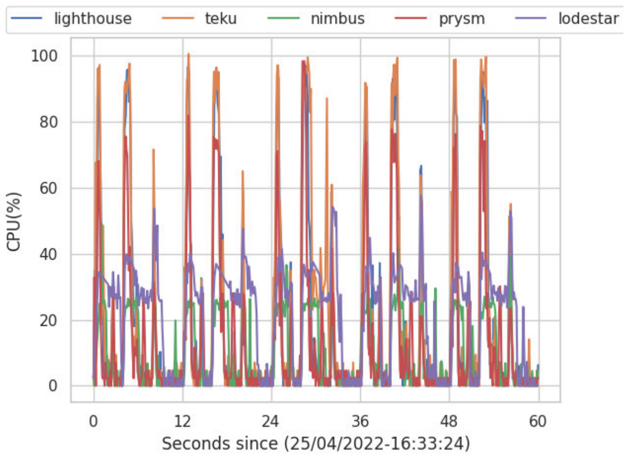

FIGURE 24. CPU utilization at the slot range while following the head of the chain.

Of course, as beacon nodes host the beacon validators, the performance, validation, and broadcasting of these duties impact the CPU workload of the nodes. Thus, we decided to measure the CPU workload of each machine with a bigger resolution, pairing it with their respective time inside the slot. Figure 24 shows the CPU workload of each client over five entire slots. Despite the workload of each client varies from the rest, the figure clearly shows three main spikes at around 4 seconds from each other. The pattern is very clear when checking the CPU utilization by Prysm, Teku, and Lighthouse, where we can appreciate that:

- The first spike corresponds to the arrival of the proposed beacon block. After one of the client's neighbours sends the block, this one has to validate the block's origin, compute all the aggregated BLS signatures of each attestation aggregation on it, and update the existing beacon state with the new block. It is known that this process is generally CPU-intense since BLS signature aggregations require complex operations like elliptic curve pairings.
- The second spike belongs to sending and receiving the corresponding attestations. Attestations or votes are small messages. However, there are many of them, and the clients actively contribute to the validation and broadcasting of the messages that they see. This explains why, on some occasions, the second spike lasts longer than the first one.
- The last spike is the least clear one. It belongs to receiving and validating the aggregated attestations that will be included in the following beacon blocks. Due to their small size and the fewer number of messages that are sent, the last spike is the smallest one.

We've seen that the CPU utilization can be easily mapped to each of the operations that the beacon node has to do. However, not all the nodes handle the workload in the same way. Prysm, Lighthouse, and Teku's behaviour has the most visual explanations of the three spikes. Nimbus and Lodestar show different profiles. They suffer smaller CPU profiles but for longer periods. It is interesting to observe how Nimbus's spikes stay under 40% of CPU utilization, keeping it for less than half a second compared with the three former clients. This smaller but longer CPU utilization means that Nimbus can handle the block arrival more efficiently, reducing the risk of not being able to process a block "fast enough" if not enough resources are available by the hosting machine. Conversely, the Lodestar behaviour is the hardest to deduce from the charts. Its CPU is used more over a more extensive period, showing that it is either not that optimized to process a new block arrival or struggles when having to share the block with the rest of the network.

B. HARDWARE RESOURCES' EVOLUTION OVER HARDFORKS

Setting up a requirements list to participate as a full Ethereum node is difficult. Previous sections IV-A IV-B IV-C already introduced the increasing hardware resource utilization tendency as the chain keeps evolving and more changes keep happening with every hardfork. This isn't something wrong; the ecosystem heavily benefits from such changes. However, this makes advising hardware that won't be obsolete in two years but without overestimating a bit more challenging, as these changes generally imply requiring more computational power, more extensive and faster disks, and heavier bandwidth usage.

Tables 6 and 7 show the mean resource utilization of each client on the different stages and hardforks of the chain. Table 6 focuses on the clients' resources while syncing the chain from the Genesis block. We can appreciate two clear patterns: i) the CPU profile and the network incoming and out-coming bandwidth remain on the same ranges for the different clients, and ii) the memory allocated by all the clients increased between 130% and 198% except for Teku that remained on the same ranges of 6.8GB to 6.9GB due to the settings on the Java Virtual Machine (JVM).

This memory-increasing pattern has many possible causes. On the one hand, the blocks and states of the beacon chain get bigger and bigger as more validators are activated in the beacon chain. This requires allocating larger items in memory than previous slots in the chain. On the other hand, with the changes introduced at Altair's hardfork, clients had to make extra computations concerning the "sync committees", computations that directly benefited by keeping more chain state information in memory.

TABLE 6. Mean resource utilization of the five main clients during their synchronization in default machines divided by Hardfork ("H.Fork"). Note: "P0" is the abbreviation for "Phase0" and "A" for "Altair", "NetIn" and "NetOut" are expressed in GB/s.

	H.Fork	CPU(%)	Mem(GB)	NetOut	NetIn
Lighthouse	P0	45.77	4.571	0.127	0.060
	A	47.42	6.031	0.127	0.050
Lodestar	P0	28.83	1.878	0.088	0.004
	A	28.78	3.724	0.088	0.004
Nimbus	P0	45.33	1.289	0.083	0.105
	A	41.44	1.676	0.082	0.028
Prysm	P0	49.96	4.636	0.043	0.010
	A	49.87	6.974	0.043	0.008
Teku	P0	62.43	6.854	0.025	0.104
	A	65.15	6.909	0.026	0.036

However, the most significant difference in resource utilization comes with the change in the status of the beacon node. Once the node is synced to the head of the chain, the number of blocks and attestations the node has to process is limited by the slots, as introduced in Section V-A. This heavily reduces the CPU utilization. Table 7 shows the mean resource utilization of the clients over five days while following the head of the chain. In the figure, we can appreciate how, in comparison to previous hard forks, the resources taken by the clients to run after Capella are considerably higher:

- the CPU utilization has reduced between 60% and 92% of the syncing values (mean decrease of 73.94%).
- the memory usage has increased between 153% and 940% (mean increase of 426%).
- the incoming network bandwidth utilization has increased between 479% and 1340% (mean increase of 824%).
- the outgoing network bandwidth utilization has increased between 838% and 9975% (mean increase of 4320%).

TABLE 7. Mean resource utilization of the five main clients during their regular operation in default machines, following the head of the chain. Note: "Cap." is the abbreviation for "Capella", "NetIn", and "NetOut" are expressed in GB/s., and the extra overhead present in the means generated from running Nethermind on each machine concurrently.

	H.Fork	CPU(%)	Mem(GB)	NetOut	NetIn
Lighthouse	Cap.	15.00	16.784	0.608	0.419
Lodestar	Cap.	11.54	17.493	0.532	0.399
Nimbus	Cap.	10.10	15.754	0.566	0.436
Prysm	Cap.	12.944	19.954	0.576	0.692
Teku	Cap.	5.37	10.60	0.262	0.030

We must note that after "the merge", the execution and consensus clients needed to be paired individually. Thus, the measurements in Table 7 aggregate both resource utilisation. This explains the sudden increase in allocated Memory and network bandwidth, as the execution client has a much larger chain state tracking the balance of all addresses and smart contracts.

In summary, hardware resource utilization is somewhat unpredictable in the long-term, as there are already new technology advances like the Distributed Validator Technology (DVT)[18] to help increase the resilience of the network or further protocol upgrades like ProtoDankSharding and DankSharding[19] that aim to help to scale the network. However, we can expect that the incoming changes will increase the hardware requirements, while still maintaining it substantially lower when compared with the previous PoW hardware evolution.

C. PERCEIVING NETWORK INSTABILITIES FROM THE CHAIN SYNCHRONIZATION

Back to the first stages of this study, when we were putting to test our methodology with the *Medalla*[20] testnet (as described in Section III-D3), we identified that syncing up the historical records of the chains could give more information than one would imagine.

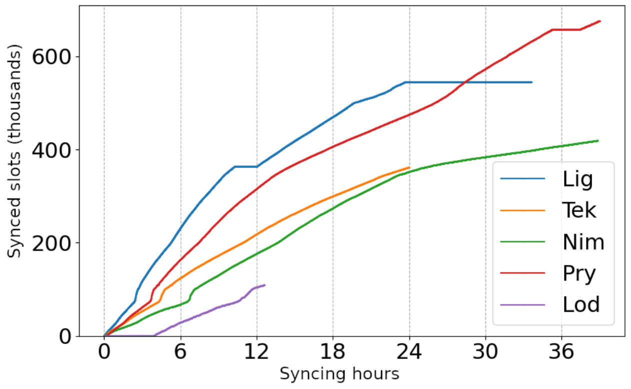

FIGURE 25. Chain synchronization time by clients in the Medalla testnet.

At the moment of syncing the testnet, the slot synchronization speed shown in Figure 25 didn't show anything relevant at first: a similar synchronization speed for most clients. At the same time, Lighthouse and Prysm were the fastest clients. However, taking a closer look at the beginning of the syncing process, we could identify a small spike of synced slots across all the clients. Zooming into the range of slots 70, 000 and 120, 000, Figure 26 showed that, in fact, the spike was constant across all the clients and not a singularity on only one of them.

Digging more into the anomaly, another metric suffered a similar distinction pattern around the same range of slots. Figure 27 shows the disk usage measured by each client. The figure shows two clear spikes in two of the controlled clients, Lighthouse and Teku, around the 2nd and 7th hour of the synchronization process. In our attempt to correlate

[18]https://ethereum.org/en/staking/dvt
[19]https://ethereum.org/en/roadmap/danksharding
[20]https://github.com/goerli/medalla

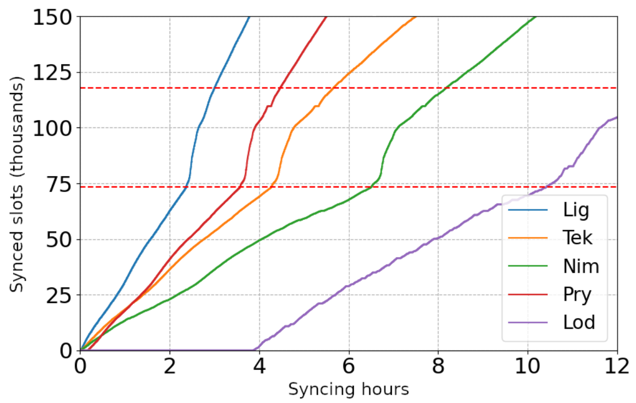

FIGURE 26. Zoomed slot synchronization speed of consensus clients in the Medalla testnet.

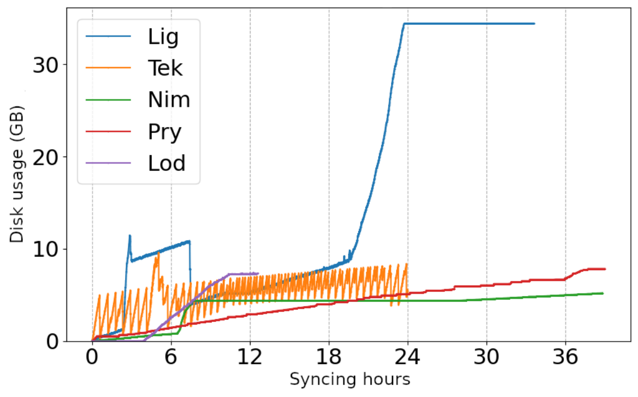

FIGURE 27. Disk usage of consensus clients during the Medalla testnet synchronization.

both anomalies, Figure 28 represents the disk usage using the synchronization slot as a reference in the X-axis. In the figure, the pattern gets more clear for all the clients except for Prysm, which barely notices any perturbance. There are several points to remark here:

- The 70.000 to 120.000 slot period corresponds with a non-finality period of the Medalla testnet which was caused by erroneous rough time responses witnessed by Prysm clients [54].
- Figure 28 shows that the disk usage reaction of Teku, Lighthouse, and Nimbus is to spike the disk usage, while Lodestar offers the exact opposite reaction, keeping the disk usage flat.

Considering the relatively similar behaviour of Lighthouse and Teku, where the sharp increase of disk usage ends with a sharp drop, it is interesting to notice how Teku reduced the time of higher disk usage, while Lighthouse keeps running for several hours with additional data before dumping it. This is due to the dual-database system that Lighthouse and some other clients use: a *hot* database that stores unfinalized beacon states, and a *cold* database that stores finalized beacon states. As they sync through a large patch of non-finality, their hot databases grow large until they reach finality and then migrate this state into the cold database.

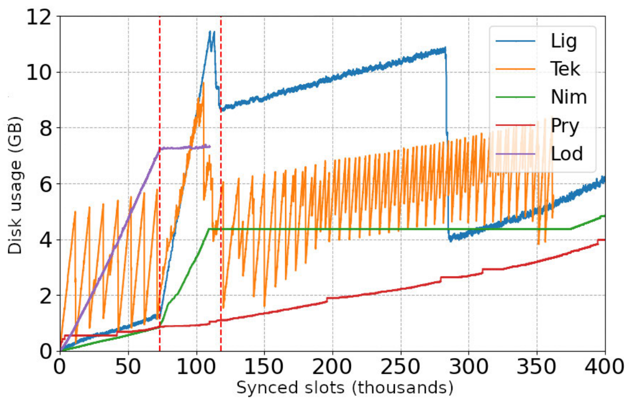

FIGURE 28. Disk Usage of Ethereum clients while syncing the Medalla testnet using the slots as X axis.

On the other hand, Nimbus's rise in disk storage is not as sharp as Teku and Lighthouse. However, its storage capacity was not reduced afterwards (unlike Teku and Lighthouse). Oddly, we can notice that Lodestar's disk usage increases more rapidly than any other client until the start of this non-finality period when it stops growing. Prysm's disk usage continues its trend without any variations as if it was not perturbed by the non-finality period. This is because Prysm clients only save finalized states every 2048 slots. This keeps disk utilization to a minimum. During non-finality, they do not save unfinalized states to disk, which allows them to prevent the database from unnecessarily growing. However, doing this comes at a cost, as they now keep everything in memory, if they need to retrieve a particular (unfinalized) state and it's been a while since finality, they have to regenerate it. Doing this puts a non-trivial amount of pressure on the CPU, making it harder to keep track of all the different forks.

The steeper syncing curve around slot 100, 000 previously seen could imply that during that time, there was little information to process (lots of missed blocks due to a lack of consensus). Therefore, clients can move faster in the syncing process. However, this does not seem to fit with the accelerating disk usage observed during the same period. To look deeper into this question, we used Lighthouse logs to analyze the number of times a block was queued and/or processed for each slot during the non-finality period. The results, depicted in Figure 29, show that during this period, there were almost no blocks queued, which seems to be consistent with the accelerated syncing speed. However, we also noticed that at the beginning of the non-finality period, at exactly slot 73, 248, 219, blocks were being queued (note the logarithmic Y axis), followed by a sudden drop of blocks to queue for more than 30, 000 slots. This clearly shows a considerable perturbation in the network.

We assume that the accelerating disk usage is related to an increase in the state stored in the client's database, which might be linked to the difficulty of pruning states during a non-finality period. Thus, to corroborate our hypothesis, we analyzed Lighthouse's detailed logs and plotted the

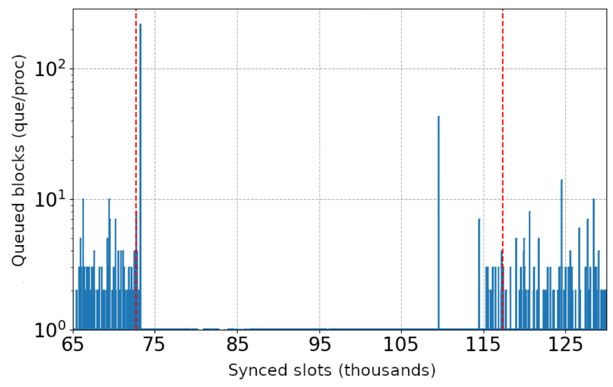

FIGURE 29. Number of times a block is queued to be persisted while syncing the Medalla chain on Lighthouse. Zoomed at the non-finality period.

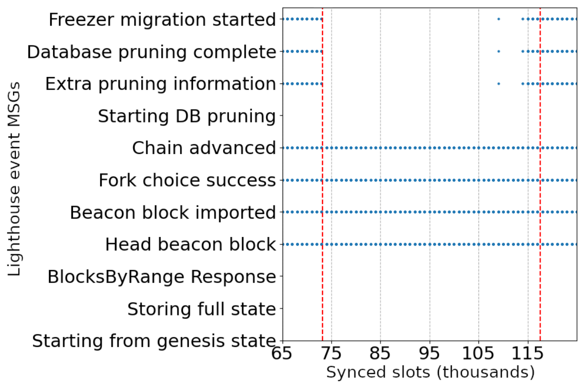

FIGURE 30. Events timeline for Lighthouse while syncing the Medalla testnet. Zoomed at the non-finality period.

frequency at which different events get executed. Figure 30 lists 11 different types of events. We can see that during the non-finality period, four types of events rarely get executed: Freezer migration started, Database pruning complete, Extra pruning information, and Starting database pruning. This demonstrates that the client could not prune the database during this period, consistent with the rapid increase in disk usage.

Although multiple things remain to be understood about the behaviour of some clients during a non-finality period, this paper demonstrates that it is possible to identify such a network disturbance by simply looking at the resource utilization of the clients.

D. BEACON API PERFORMANCE

Access to chain data or internal node information is crucial for most users. Thus, we performed two sets of experiments that benchmark the REST APIs of the different clients following the methodology described in III-E2. The first one consisted of 1.000 sequential queries performed with an in-between delay of 10 seconds (to prevent the exhaustion of the resources of the beacon node). Table 8 shows the success ratio of each client supporting the archival mode, defining the reliability of each client under the same workload. The low success of Prysm catches our attention for having the lowest percentage of successfully replied queries, a 19%. Digging into the possible root of the problem for such a low success rate, we discover that Prysm synced until the last slot we added to our query randomizer. However, by checking the response times in Figure 31, the successful calls didn't perform that well compared with the rest of the clients. Prysm is known for being the only client that offers endpoints for both gRPCs and HTTP REST API calls. They strongly believe that gRPCs are better and faster. Even though this is a legitimate statement, at Prysm's beacon node level, the REST API calls are translated into gRPC on their arrival and vice-versa when returning the response, making the process slower than other clients. Given that the rest of the beacon nodes communicate with their validator client through the REST API, it leaves Prysm in a lower step towards client interoperability. Checking the rest of the time responses, we do remark that Teku managed to reply to all the queries in under 10 seconds, with a median of 1.23 seconds.

TABLE 8. Percentage of responses the Beacon node could successfully reply under different concurrent request workloads.

Concurrent Queries	Prysm	Lighthouse	Teku	Nimbus
1	19%	100%	92%	98%
10	0%	93%	99%	0%

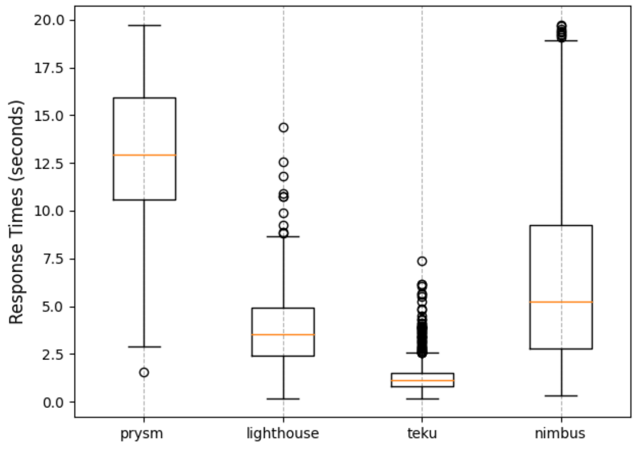

FIGURE 31. Distribution of the Beacon node's response time under a single concurrent request workload. Note that 20 seconds was the timeout of the request.

In the second experiment, we increased the number of performed concurrent queries from one to ten, checking each client's limits. The second row of Table 8 shows that only Teku and Lighthouse kept a similar reliability ratio at such a level of demand, keeping a 90% of successful replies. In comparison, Prysm and Nimbus stayed far behind with a 0%. Regarding the response times of the ten concurrent queries shown in Figure 32, most of the response times distributions got higher and larger. Lighthose's and Prysm's response times got more concentrated around the predefined *Timeout* of 20 seconds. As expected, Lighthouse's median moved from around 3.1 seconds to 13 seconds, while Teku's median response time increased up to 4.2 seconds.

The clear difference between the client's throughput and response times is easily attributable to the different end-user targets. While Prysm and Nimbus might be performance-focused, Lighthouse and Teku have a larger business-research-oriented focus, where Lighthouse offers a larger set of API endpoints for data accessibility, and Teku was built with the idea of healing many requests coming from Infura[21].

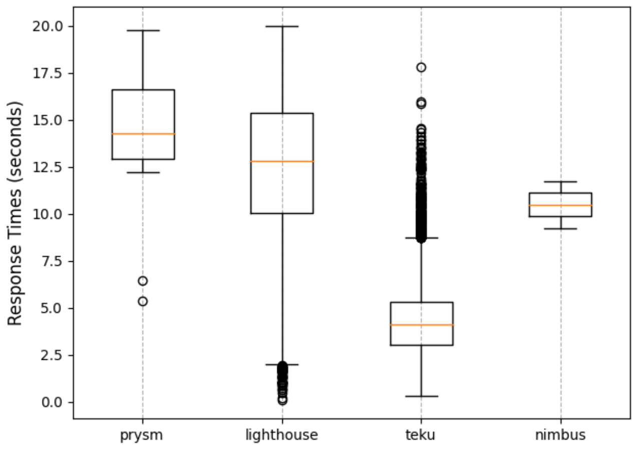

FIGURE 32. Distribution of the Beacon node's response time under ten concurrent requests workload. Note that 20 seconds was the timeout of the request.

VI. DISCUSSION

In our journey through this exhaustive evaluation, we found multiple strong points as well as room for improvement in all Ethereum CL clients. Our objective with this study was fourfold: i) introduce the different implementations available to participate in the network, with their respective strong and weak points; ii) dissect the necessary hardware resources on each of the stages of an Ethereum node (syncing vs following the head) and provide an overview of how the network can affect the available resources; and last but not least, iii) introduce how the network and the ecosystem can directly benefit from individual choices such as choosing client or home-staking.

A. STRENGTHS AND WEAKNESSES OF THE CLIENTS

The evaluation presented above gives empirical data about how the Ethereum CL clients perform under different hardware configurations and network scenarios. However, many other aspects also play an important role when choosing the software that will be deployed on an operational platform, such as documentation of the clients' usage, functionalities of the exposed API (important if the client will be used as an entry point to Ethereum's on-chain data). Although some of these aspects might be subjective, we try to cover some of those aspects together with the empirical data measured, discussing the strengths and points for improvements of each Ethereum CL client.

[21]https://beaconcha.in/

1) PRYSM

Prysm has one of the best user experiences among its clients. It is easy to set up and deploy on its default configuration. The Prysmatic Labs[22] team has done a remarkable job simplifying the deployment for non-technical users. However, on the other hand, it has room for optimization in several aspects. The documentation portal could be improved; finding information on configuring certain parameters is not intuitive. The API offered by the client could be highly improved. The API is generally used as a communication point between the validator and the beacon node using gRPCs. Despite gRPC being a nice alternative to the standard HTTP endpoint APIs, Prysm has to comply with the standard HTTP API that allows interoperability with other validator clients. And because they rely on gRPC by default, the performance of the HTTP endpoints gets massively impacted as each HTTP request has to be translated to gRPCs. The result of this performance impact is a slower API that can't support multiple requests simultaneously. The synchronization of the client in an archival mode (storing the beacon state checkpoints with a very low frequency for faster access to them) is also slower when compared with other clients.

2) LIGHTHOUSE

Lighthouse from Sigma Prime[23] was the client with the most complete API. It has all the CL Beacon node API standard implemented, and they have extended the endpoints with others that we found interesting from a research or data analysis point of view. On the other side, the client on the first measurements had a memory leak while syncing the chain from Genesis, it also has the highest disk requirements among all clients and seems to be among the most chatty clients. As with Prysm, the number of disk IO management should be reviewed, as other clients have shown that it can be considerably reduced.

3) TEKU

Teku seems to be one of the most stable clients with very complete documentation, in which it is really easy to find any execution option and command line flag. It has a very competent archival node mode, as Consensys[24] uses it to provide all the information through Infura[25]. However, generally, its synchronization is pretty slow, and when it comes to the archival node (which definitely needs to be synced from Genesis), it takes a lot more space than comparing it with others. On the other side, its standard client has the lowest storage needs, and the archival mode has the fastest API response time of all clients. It is also really important to note that setting up the JVM correctly to avoid memory issues can be tricky at the beginning, so it might take a few tries to properly set it up.

[22]https://prysmaticlabs.com/
[23]https://sigmaprime.io/
[24]https://consensys.io/
[25]https://www.infura.io/

4) NIMBUS

Nimbus from Status[26] is the client with the lowest CPU and memory requirements across all platforms and the fastest syncing open-source client. It is clearly the client better suited to run on low-power devices, but it also performs well on more powerful servers. On the other hand, its compilation and deployment are not as user-friendly as other clients. Also, the fact that the Beacon node and Validator node run on the same executable could be viewed as a feature but also as a disadvantage, as sometimes it is useful to stop the Validator client while keeping the Beacon node alive.

5) LODESTAR

Lodestar from ChainSafe[27] is one of the latest CL clients to join the race, and it is commendable to see that the software supports most of the features that the other clients offer. Also, it shows a fairly low resource consumption. However, Lodestar is not always easy to compile and deploy (except when using Docker), there were multiple outdated instructions in the documentation. It is also the slowest client to sync from Genesis, and it does not offer archival mode.

6) GRANDINE

Grandine was the fastest client to sync across all. It seems to have a great parallelization strategy that outperforms other clients while syncing from Genesis. However, it is not sure how much this speed can impact the performance after syncing. Despite there are still many features in beta, clearly, the biggest drawback of this client is that it has not been open-sourced yet.

B. HARDWARE RECOMENDATIONS

We have shown that the running combo CL plus EL clients is way more memory-demanding under standard conditions, which involves clients following up with the canonical head of a finalizing chain. However, we do see that in the moments of having to re-sync some part of the chain, either because there was a chain re-organization when some blocks didn't get enough votes and had to be dropped, forcing to resync the last finalized checkpoint or simply because the client went down, the CPU spikes 3-4 times that usage.

Similar resource spikes are expected if the network is experiencing difficulties in finalizing, where more uncertain states and blocks must be kept in the disk and memory. So having that extra hardware, CPU, memory, and faster and bigger disks can make the difference between recovering faster from this rare behaviour of the chain or being penalized by it because the client struggles to recompute and sync up the correct head of the chain because CPU, memory, or disk are at their 100% capacity, but it is insufficient.

This shouldn't be extrapolated to choose renting the "fattest" or "biggest" possible machine to run a node, as we support a middle ground where hardware should be slightly overestimated to satisfy those sudden need spikes. However, we do believe that the current requirements and performance of the clients exceed the hardware of domestic low-performance hardware devices such as a *Raspberry Pi*. Of course, some tweaks and upgrades can be done in the hardware of Raspberry Pis, like extending the disk speeds with external SSDs, buying more powerful modifications or alternatives, upgrading it with extension packs, and so on, but compatibility and the community easily troubleshoot complications might leave this option to enthusiasts.

At the time of writing this paper, we found a sweet spot for the hardware on machines with 8 CPU cores, preferably 32GB of memory, and a fast 2TB SSD disk. This is a good result since many PCs can satisfy these requirements. There are solid low-powered devices such as laptops, Inte-NUCS, Mac Minis, and so on that could run an Ethereum node without any problems and the need to build a custom PC. On the other hand, there are custom solutions like DappNode machines that can help and guide less experienced or technical users to maintain their nodes, which summarizes the configuration and maintenance of the node in a few clicks.

C. THE NETWORK CAN BENEFIT FROM YOUR CLIENT CHOICES

We have already presented and discussed the different requirements of the different available CL clients. However, they all have shown to be reliable, and there isn't much difference between them that makes any of them a clearer or better choice. We believe each of them has its target users, but we have to encourage users to try them all out and choose the one they feel more comfortable with. Client diversity is an important aspect of the network's resilience and ultimately the chain. So exploring the least popular client choices can not only help the community but also surprise us with a better performance than the one we expected.

Following the same line of recommendations, we highly encourage users to lose the fear of playing around with spawning their own nodes and to stake from home. These are very good practices that help the decentralization and resilience of the entire network. Furthermore, there is a broad literature, forums, communication channels, and a friendly community willing to support setting up or troubleshooting the spin-up of Ethereum nodes.

VII. CONCLUSION

In this paper, we have shown multiple aspects of all Ethereum CL clients while tested under different conditions. We have exposed their strengths and discussed some points for improvement. After all these experiments, it seems clear that the different CL client teams have focused on different aspects, users, and use cases and excel in different points.

Perhaps the most important conclusion that should be highlighted is that all Ethereum CL clients run well on different hardware platforms and configurations. They showcase Ethereum's strong software diversity, which is hard to find anywhere else in the blockchain ecosystem. Overall,

[26]https://status.im/
[27]https://chainsafe.io/

our evaluation demonstrates that the efforts of all CL client implementation teams and researchers involved have pushed the Ethereum ecosystem one step closer to a more sustainable and scalable blockchain technology.

APPENDIX
BEACON CHAIN SYNCHRONIZATION ON A RASPBERRY PI 4

We have run all the different clients from scratch in Raspberry Pis on the versions displayed in section III-D, testing if running a production node in such a low-powered device is possible. Anticipating the slowest performance of the more limited resources of the Raspberry Pis, the syncing measurement of the chain for the low-powered devices was performed on the *Kiln* testnet[28] on the same range of dates (18th of March, 2022).

1) CPU UTILIZATION

Figure 34 shows the CPU utilization degree achieved by the CL clients, which remains relatively similar to the one measured on the default node with a slight overhead as each core is ‘‘slower’’ than its *default* version. The figure shows that Grandine increases CPU utilization from 80% to a constant 90% CPU usage while syncing. Teku follows the lead, increasing its CPU utilization up to the 80% as the blocks keep getting bigger due to the aggregation of more validators. At the same time, the rest of the clients also seem to register an increase of the 10% of the usage.

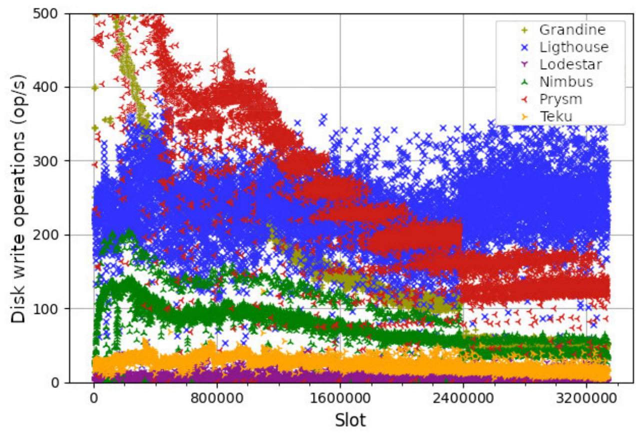

FIGURE 33. Disk write operations by the clients while running in a “Fat” node.

2) MEMORY UTILIZATION

A similar pattern was observed in terms of memory. Figure 35 shows the memory allocation patterns from the CL clients in the Raspberry Pis. Teku keeps a steady performance by assigning 6*GB* of memory to the JVM. Prysm keeps allocating memory as *Go* does not free memory unless the OS asks for it, reaching 5*GB* of memory in its latest synchronization stages. Lodestar, Grandine, and Nimbus

[28]https://github.com/eth-clients/merge-testnets/tree/main/kiln

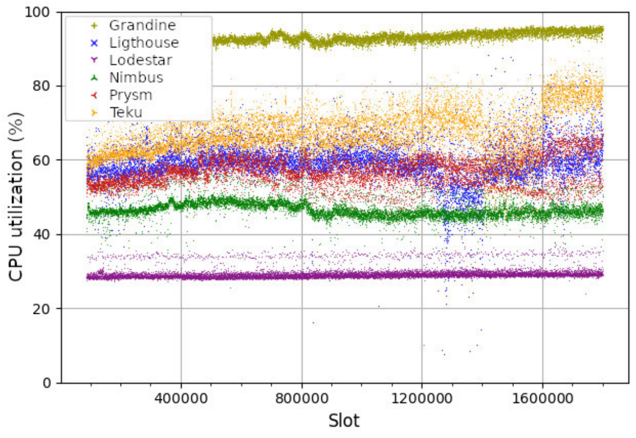

FIGURE 34. CPU utilization while syncing the chain in a Raspberry Pi 4.

remain with the lower profile at a lower limit of 3*GB*, showing that not much memory is needed to sync up the chain. Finally, Lighthouse shows the same memory leak pattern but sooner this time, as the machine’s total memory is reduced to 8*GB*. We experienced three client crashes as more memory than the available was asked. In this sense, each sudden drop in Figure 35 belongs to each of the restarts of the client after the crash. As Figure 36 shows, with the shorter access to memory resources, the disk utilization remains steady from the beginning of the synchronization process as opposed to the previous hardware configurations. However, not all the clients have the same disk usage. In the figure, we can see that in synchrony with its memory leak, Lighthouse highly relies on disk write operations in comparison to other clients, multiplying by three times the usage of Teku, and by more than eight times the rest.

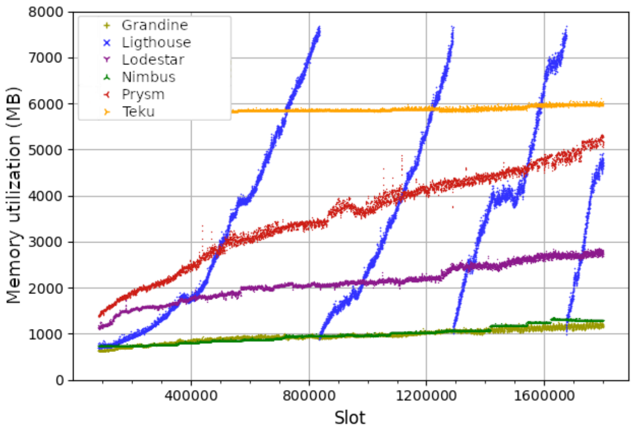

FIGURE 35. Memory utilization while syncing the chain in a Raspberry Pi 4.

3) PERFORMANCE

In the opposite direction as the one measured with the ‘‘fat nodes’’, Figure 15 shows decreasing the hardware resource clearly impacts syncing the chain. Making it clear, once again, that being slower while syncing the chain shouldn’t be a

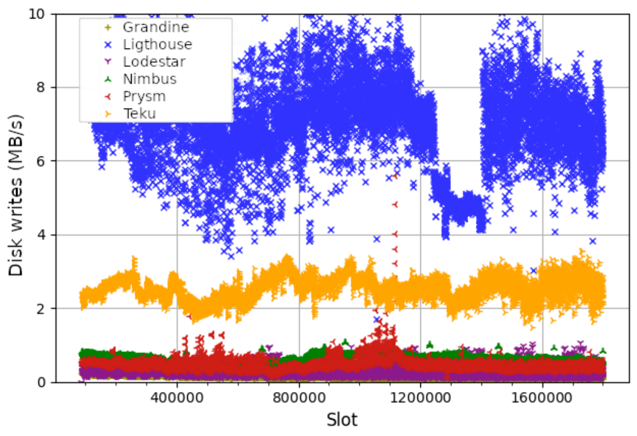

FIGURE 36. Disk writing speeds while syncing the chain in a Raspberry Pi 4.

determinant factor, as syncing via checkpoints significantly reduces the overall duration of this process. In any case, slower CPU cores, less memory, and slower reading and writing speeds from and to SD cards have drawbacks and impact performance.

ACKNOWLEDGMENT
The authors would like to thank the client developer teams for their support in troubleshooting different encountered issues and their feedback on this work and also would like to thank Paristosh from the Ethereum Foundation for his involvement and support in the project and Izzy for his constructive feedback on this study.

REFERENCES

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.

[2] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu, and G. Rosu, "KEVM: A complete formal semantics of the Ethereum virtual machine," in *Proc. IEEE 31st Comput. Secur. Found. Symp. (CSF)*, Jul. 2018, pp. 204–217.

[3] M. Wohrer and U. Zdun, "Smart contracts: Security patterns in the Ethereum ecosystem and solidity," in *Proc. Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Mar. 2018, pp. 2–8.

[4] M. Cortes-Goicoechea and L. Bautista-Gomez, "Discovering the Ethereum2 P2P network," in *Proc. 3rd Conf. Blockchain Res. Appl. Innov. Netw. Services (BRAINS)*, Sep. 2021, pp. 1–2.

[5] P. Henrique F. S. Oliveira, D. Müller Rezende, H. S. Bernardino, S. M. Villela, and A. B. Vieira, "Analysis of account behaviors in Ethereum during an economic impact event," 2022, *arXiv:2206.11846*.

[6] S. Tikhomirov, "Ethereum: State of knowledge and research perspectives," in *Proc. Int. Symp. Found. Pract. Secur.* Cham, Switzerland: Springer, 2017, pp. 206–221.

[7] S. Nakamoto, "Bitcoin," in *A Peer-to-Peer Electronic Cash System*, vol. 21260, 2009.

[8] A. Hayes, "A cost of production model for Bitcoin," Tech. Rep., 2015.

[9] F. Saleh, "Blockchain without waste: Proof-of-stake," *Rev. Financial Stud.*, vol. 34, no. 3, pp. 1156–1190, Feb. 2021.

[10] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and casper," 2020, *arXiv:2003.03052*.

[11] O. Moindrot and C. Bournhonesque, "Proof of stake made simple with casper," in *Proc. ICME*, 2017.

[12] D. Park, Y. Zhang, and G. Rosu, "End-to-end formal verification of Ethereum 2.0 deposit smart contract," in *Proc. Int. Conf. Comput. Aided Verification*, Los Angeles, CA, USA. Cham, Switzerland: Springers, 2020, pp. 151–164.

[13] D. Sel, K. Zhang, and H.-A. Jacobsen, "Towards solving the data availability problem for sharded Ethereum," in *Proc. 2nd Workshop Scalable Resilient Infrastructures Distrib. Ledgers*, Dec. 2018, pp. 25–30.

[14] M. Hall-Andersen, M. Simkin, and B. Wagner, "Foundations of data availability sampling," *Cryptol. ePrint Arch.*, 2023.

[15] V. Denisova, A. Mikhaylov, and E. Lopatin, "Blockchain infrastructure and growth of global power consumption," *Int. J. Energy Econ. Policy*, vol. 9, no. 4, pp. 22–29, Jul. 2019.

[16] A. O. Mahony and E. Popovici, "A systematic review of blockchain hardware acceleration architectures," in *Proc. 30th Irish Signals Syst. Conf. (ISSC)*, Jun. 2019, pp. 1–6.

[17] J. Li, N. Li, J. Peng, H. Cui, and Z. Wu, "Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies," *Energy*, vol. 168, pp. 160–168, Feb. 2019.

[18] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 3–16.

[19] A. Pinar Ozisik, G. Bissias, and B. Levine, "Estimation of miner hash rates and consensus on blockchains (draft)," 2017, *arXiv:1707.00082*.

[20] M. Bedford Taylor, "The evolution of Bitcoin hardware," *Computer*, vol. 50, no. 9, pp. 58–66, 2017.

[21] L. Wang and Y. Liu, "Exploring miner evolution in Bitcoin network," in *Proc. Int. Conf. Passive Act. Netw. Meas.*, New York, NY, USA. Cham, Switzerland: Springer, 2015, pp. 290–302.

[22] M. Sivanesan, A. Chattopadhyay, and R. Bajaj, "Accelerating hash computations through efficient instruction-set customisation," in *Proc. 31st Int. Conf. VLSI Design 17th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2018, pp. 362–367.

[23] Z. A. Al-Odat, M. Ali, A. Abbas, and S. U. Khan, "Secure hash algorithms and the corresponding FPGA optimization techniques," *ACM Comput. Surveys*, vol. 53, no. 5, pp. 1–36, Sep. 2021.

[24] N. Houy, "The Bitcoin mining game," *Ledger*, vol. 1, pp. 53–68, Dec. 2014.

[25] J. Anish Dev, "Bitcoin mining acceleration and performance quantification," in *Proc. IEEE 27th Can. Conf. Electr. Comput. Eng. (CCECE)*, May 2014, pp. 1–6.

[26] J. Shen, A. L. Varbanescu, and H. Sips, "Look before you leap: Using the right hardware resources to accelerate applications," in *Proc. IEEE IEEE Int. Conf. High Perform. Comput. Commun. 6th Int. Symp. Cyberspace Saf. Secur. 11th Int. Conf. Embedded Softw. Syst.*, Aug. 2014, pp. 383–391.

[27] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on Ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surveys*, vol. 53, no. 3, pp. 1–43, May 2021.

[28] P. Silva, D. Vavricka, J. Barreto, and M. Matos, "Impact of geo-distribution and mining pools on blockchains: A study of Ethereum," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2020, pp. 245–252.

[29] M. Cortes-Goicoechea, T. Mohandas-Daryanani, J. L. Munoz-Tapia, and L. Bautista-Gomez, "Unveiling Ethereum's hidden centralization incentives: Does connectivity impact performance?" 2023, *arXiv:2309.13329*.

[30] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring Ethereum network peers," in *Proc. Internet Meas. Conf.*, Oct. 2018, pp. 91–104.

[31] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, Nov. 2018.

[32] P. K. Sharma, N. Kumar, and J. H. Park, "Blockchain technology toward green IoT: Opportunities and challenges," *IEEE Netw.*, vol. 34, no. 4, pp. 263–269, Jul. 2020.

[33] I. Romashkova, M. Komarov, and A. Ometov, "Demystifying blockchain technology for resource-constrained IoT devices: Parameters, challenges and future perspective," *IEEE Access*, vol. 9, pp. 129264–129277, 2021.

[34] Y. Gao, J. Shi, X. Wang, Q. Tan, C. Zhao, and Z. Yin, "Topology measurement and analysis on Ethereum P2P network," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2019, pp. 1–7.

[35] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *Proc. Eur. Symp. Res. Comput. Secur.*, Sophia Antipolis, France. Cham, Switzerland: Springer, 2004, pp. 423–438.

[36] A. Nguyen-Tuong, D. Evans, J. C. Knight, B. Cox, and J. W. Davidson, "Security through redundant data diversity," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. FTCS DCC (DSN)*, 2008, pp. 187–196.

[37] B. Littlewood, P. Popov, and L. Strigini, "Modeling software design diversity: A review," *ACM Comput. Surv.*, vol. 33, no. 2, pp. 177–208, Jun. 2001.

[38] P. Traverse, "AIRBUS and ATR system architecture and specification," in *Software Diversity in Computerized Control Systems*. Cham, Switzerland: Springer, 1988, pp. 95–104.

[39] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese, "Network diversity: A security metric for evaluating the resilience of networks against zero-day attacks," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 5, pp. 1071–1086, May 2016.

[40] D. Borbor, L. Wang, S. Jajodia, and A. Singhal, "Optimizing the network diversity to improve the resilience of networks against unknown attacks," *Comput. Commun.*, vol. 145, pp. 96–112, Sep. 2019.

[41] P. Reibel, H. Yousaf, and S. Meiklejohn, "Short paper: An exploration of code diversity in the cryptocurrency landscape," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2019, pp. 73–83.

[42] T. Inagaki, Y. Ueda, T. Nakaike, and M. Ohara, "Profile-based detection of layered bottlenecks," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, Apr. 2019, pp. 197–208.

[43] A. Noureddine, R. Rouvoy, and L. Seinturier, "Monitoring energy hotspots in software: Energy profiling of software code," *Automated Softw. Eng.*, vol. 22, no. 3, pp. 291–332, Sep. 2015.

[44] M. Cortes-Goicoechea, T. Mohandas-Daryanani, J. Luis Muñoz-Tapia, and L. Bautista-Gomez, "Autopsy of Ethereum's post-merge reward system," 2023, *arXiv:2303.09850*.

[45] S. Rouhani and R. Deters, "Performance analysis of Ethereum transactions in private blockchain," in *Proc. 8th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2017, pp. 70–74.

[46] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "GossipSub: Attack-resilient message propagation in the filecoin and ETH2.0 networks," 2020, *arXiv:2007.02754*.

[47] *Node Exporter*. [Online]. Available: https://github.com/prometheus/node_exporter

[48] *Prometheus*. [Online]. Available: https://github.com/prometheus/prometheus

[49] *Ethereum Beacon Node Rest Api Standard*. [Online]. Available: https://ethereum.github.io/beacon-APIs/

[50] *Ethereum Execution API Standard*. [Online]. Available: https://github.com/ethereum/execution-apis

[51] *API Benchmark Tool*. [Online]. Available: https://github.com/cortze/api-benchmark

[52] M. Cortes Goicoechea, T. Mohandas-Daryanani, J. L. Muñoz-Tapia, and L. Bautista-Gomez. *Miga Labs' Ethereum Client's Hardware Resource Analysis*. [Online]. Available: https://github.com/migalabs/eth-client-hw-analysis

[53] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2017, *arXiv:1710.09437*.

[54] *Medalla Non-finality Period August 2020*. [Online]. Available: https://docs.google.com/document/d/11RmitNRui10LcLCyoXY6B1IN CZZKq30gEU6BEg3EWfk

MIKEL CORTES-GOICOECHEA received the bachelor's degree in industrial electronics and the master's degree in embedded systems from the University of the Basque Country, Spain. He is currently pursuing the Ph.D. degree with Universidad Politécnica de Catalunya. He is also a Research Engineer with the Barcelona Supercomputing Center and MigaLabs, Spain. His research interests include p2p networks, p2p protocols, and blockchain applications, such as Ethereum, IPFS, and Filecoin. He was awarded with a Ph.D. Research Fellowship from Protocol Labs, during which he could collaborate on various research projects with entities, such as Protocol Labs, the Filecoin Foundation, the Ethereum Foundation, the University of Cambridge, and the Codex Storage Team at Status.

TARUN MOHANDAS-DARYANANI was born in Canary Islands, Spain, in 1996. He received the degree in computer science from the University of La Laguna, Canary Islands, and the master's degree in information security technology from the Open University of Catalonia. From 2018 to 2019, he participated in a scholarship with Telefonica, where he developed several tools to automate the data transfer between APIs and maintained some of the video platform infrastructure systems. From 2019 to 2021, he was a Cybersecurity Consultant with Daimler Group Services Madrid, where his main role was reviewing the software development lifecycle of the internal applications. Since 2021, he has been a Research Engineer with Migalabs, a small research group on the Ethereum blockchain, where he has developed both code and infrastructure. Among his main experiments and research, we can highlight some performance analyses: the resource analysis on the Ethereum consensus clients and the distributed validator performance analysis in collaboration with the Obol Team. He is also the main maintainer of the GotEth tool, a lightweight beacon chain data extractor that powers the Ethseer website.

JOSE LUIS MUÑOZ-TAPIA received the M.S. degree in telecommunications engineering, in 1999, and the Ph.D. degree in security engineering, in 2003. He is currently a Researcher with the Information Security Group, an Associate Professor with the Department of Network Engineering, Universidad Politécnica de Catalunya (UPC), and the Director of the Master Program in Blockchain Technologies with the UPC School. His research interests include applied cryptography, network security, game theory models applied to networks and simulators, and distributed ledgers technologies.

LEONARDO BAUTISTA-GOMEZ received the master's degree from Pierre & Marie Curie Paris 6 University and the Ph.D. degree from Tokyo Institute of Technology. He is currently the Founder and a Team Leader of MigaLabs and he is also a Senior Researcher with the Codex Team. He has been collaborating with the EF for more than five years and he has received five research grants from the EF, plus several research grants from Lido, Obol, and other institutions. He has over a decade of research experience in supercomputers, deep learning, and blockchain technology. He has published more than 50 scientific articles and he has received multiple international academic awards, such as the IEEE TCSC Award for Excellence in Scalable Computing and the ACM/IEEE George Michael Memorial High Performance Computing Fellow.