Resource Analysis of Ethereum 2.0 Clients

Mikel Cortes-Goicoechea
Barcelona Supercomputing Center
Barcelona, Spain
mikel.cortes@bsc.es

Luca Franceschini
Barcelona Supercomputing Center
Barcelona, Spain
luca.franceschini@bsc.es

Leonardo Bautista-Gomez
Barcelona Supercomputing Center
Barcelona, Spain
leonardo.bautista@bsc.es

***Abstract*—Scalability is a common issue among the most used permissionless blockchains, and several approaches have been proposed to solve this issue. Tackling scalability while preserving the security and decentralization of the network is an important challenge. To deliver effective scaling solutions, Ethereum is on the path of a major protocol improvement called Ethereum 2.0 (Eth2), which implements sharding. As the change of consensus mechanism is an extremely delicate matter, this improvement will be achieved through different phases, the first of which is the implementation of the Beacon Chain. For this, a specification has been developed, and multiple groups have implemented clients to run the new protocol. This work analyzes the resource usage behavior of different clients running as Eth2 nodes, comparing their performance and analyzing differences. Our results show multiple important network perturbations and how different clients react to them. We discuss the differences between Eth2 clients and their limitations.**

***Index Terms*—Blokchain, Ethereum2, Eth2, Beacon Chain, Sharding, Clients, Proof of Stake, Smart Contracts, Scaling**

I. Introduction

Ethereum [1] has been a great achievement in the road to ubiquitous blockchain technology. It led to a huge growth in the number of decentralized applications due to its general-purpose virtual machine and its dedicated programming language. These characteristics have set the conditions for a solid community of developers and continuous advancements as well as introducing new technological possibilities. As the Ethereum adoption increases, its usability has been threatened by the rising transaction volume and network clogging.

To successfully implement effective scaling solutions, Ethereum is on the path to a major protocol improvement that will enhance its scalability by several orders of magnitude and provide an architecture to flexibly address the needs of a constantly changing industry. Ethereum 2.0 (Eth2) is based around the concept of sharding, where the blockchain is split into shards, and subsets of validators are randomly assigned to each shard to validate transactions. As validators just need to validate transactions relative to the shards they have been assigned to, parallelism and, therefore, scalability is achieved. To have a coherent state of the network between different shards, the roots of the shard blocks are appended to a dedicated chain called the Beacon Chain. This Beacon Chain is considered to be the *heart beat* and the foundation of the sharding protocol. Validators shuffle randomly every epoch, verifying transactions on different shards.

To embrace these changes to the final stage of Eth2, the consensus mechanism of Ethereum will change from Proof-of-Work (PoW) to Proof-of-Stake (PoS). This change allows blocks to be produced more sustainably, saving electricity while implementing a more exhaustive network infrastructure. Due to the complexity of introducing this consensus mechanism, the achievement of sharding in Eth2 has been split into different phases. In the first phase, called ”Phase 0” [2], PoS is implemented on the Beacon Chain and validators participate as part of proposers and attestation committees. Based on the previous state of the chain and on the balances, a randomly chosen validator has a time window of 12 seconds to propose a new block. This time window, also known as slot, is when other committee participants perform their attestations. If a validator does not perform its duties, it gets exposed to economic slashing, while honesty and participation get rewarded.

As the last official testnet before the real launch of the Beacon Chain, the *Medalla* testnet was launched on the 4th of August by the Ethereum community. The purpose of the test net was to evaluate the stability and reliability of the implementation that the five main developer teams generated. Given the valueless economic tokens that were used to participate in the testnet, this one got exposed to all kinds of events. The experienced events went from massive validator disconnections to attack attempts to prevent the finalization of the chain, bringing the implementations and the protocols to the limits.

In this paper, we aim to compare the performance and the resources needed by the five main Eth2 clients available. We will also dig into several network perturbations that occurred in the medalla testnet and how different clients reacted to it.

The remainder of this paper is organized as follows. Section III explains the methodology used for the evaluation. In Section IV we show and analyze the results obtained by our study. Section II discusses related work. Finally, Section VI concludes this work and presents some future directions.

II. Related Work

In recent years, there have been significant efforts to study the scalability and security of the Ethereum network from multiple perspectives. The importance of a new consensus mechanism for Ethereum can be understood through the scalability challenges that the protocol faces [3]. Peers' participation in the Ethereum network has been studied previously [4] from a p2p network perspective unveiling the Ethereum network topology.

Eth2 clients rely on the Proof-of-Stake (PoS) consensus mechanism called Casper [5] and some works have shown insights on Casper's robustness depending on the network latency [6]. However, Casper and the Beacon Chain are just the parts of a fully sharded Eth2 protocol, and its following phase concerns the problem of data availability [7]. In addition, the current security threats of Ethereum have also been analyzed recently [8]. While all these studies are precious for the community, they all focus on the protocol security aspects, and none of them evaluates the actual implementations of the Eth2 protocol.

Eth2 consensus mechanism requires validators to lock ETH into a smart deposit contract, whose security is essential. In order to verify and increase the safety of the deposit contract, formal verification of the Eth2 deposit contract has been done [9]. This study does verify the implementation of the deposit contract but not the Eth2 clients. While Eth1 clients have also been analyzed in the past [10], this is the first study showing the resource utilization of Eth2 clients and what can be understood from it.

III. Methodology

To study differences in the behavior of Eth2 clients, we have been running the Eth2 clients Teku, Nimbus, Lighthouse, Prysm, and Lodestar for several hours, letting the clients sync to the Eth2 chain from scratch. The objective of this study is to monitor specific metrics in order to understand the behavior and performance of the clients while syncing to the Eth2 network. In addition to some differences in clients' behavior that may have been caused by conditions of the Medalla Test Network [11] at a given time, we have observed some patterns and gathered some insights about the clients. The metrics that have been monitored are:

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

The metrics mentioned above have been collected by launching the clients with an empty database and no external processes running. After having located the client process ID, we launched a resource monitoring script written in python [12] that records the resource utilization every second into a data file. While launching the clients, a flag has been added in order to save clients' logs to a text file. The logs were parsed through python scripts [13], allowing to extrapolate the syncing speed of the clients. Metrics and slot times were plotted through python-matplotlib scripts [14], and saved into CSV files.

A. Evaluation platform

Tests have been run on two nodes, each node with an Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz with 1 core and two threads. Each client has been run on a single node with 4.8 GB of total memory available and 34GB of storage (40GB total, of which 6GB is used for the operating system).

Clients have been run on the Medalla Test Network using their default configuration. No additional flags or configurations have been used, with the exception of a flag on Teku that limits the JVM memory and raises the heap size of Lodestar on the package.json. Teku ran allocating 2 GB to the JVM, as allocating more than 2GB would make the server crash. Similar issues occurred with Lodestar.

B. Client versions and running period

Clients have been run with the following versions:

- Teku: v0.12.14-dev-6883451c [15]
- Prysm: v1.0.0-beta.1-4bc7cb6959a1ea5b [16]
- Lighthouse: v0.3.0-95c96ac5 [17]
- Nimbus: 0.5.0-9255945f [18]
- Lodestar: commit 40a561483119c14751 [19]

Clients have been running during the periods depicted in Table I. Teku and Lighthouse were run several times in order to compare the resource usage of two different executions.

TABLE I: Client Running Periods

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

IV. Analysis

In this section we go over the results obtained during the runs of the five clients and we analyse their differences in resource consumption and behaviour.

A. Client syncing

First, we start by studying the synchronization time of all clients. Note that the synchronization time of a client is not the most determining factor of its efficiency. First, this is a cost that it should be paid only once in the lifetime of a node (assuming no hard failures that force a restart from scratch), but also because most clients can now start from a weak subjectivity state and be up and running in under a minute. Nonetheless, synchronization time is relevant when comparing it with other resources metrics to try to uncover hidden effects or understand unexpected behaviour and this is the way we use it in this research. Other studies have also used syncing metrics in a similar way [20].

Figure 1 shows the synchronization slot as the client runs from genesis. We tried to run all clients for over 24 hours, with the exception of Lodestar, which suffered from multiple crashes and had to be restarted several times. We can see that Lighthouse appears to be the fastest syncing client, though its syncing process stops around hour 24 due to reaching the maximum storage capacity of the server (See Section IV-G). After Lighthouse stopped its syncing process, Prysm was able

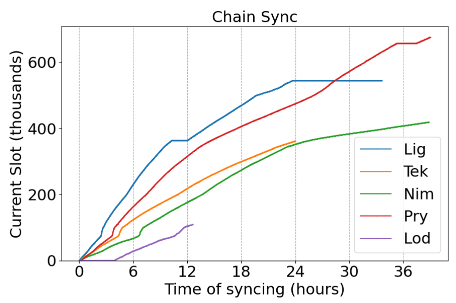

Fig. 1: Slot synchronization of Eth2 clients

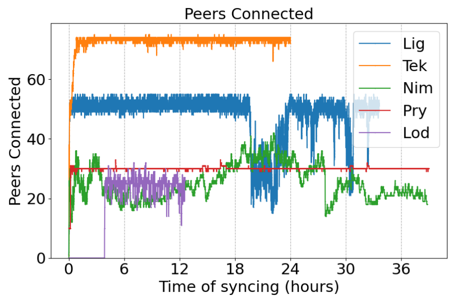

Fig. 2: Peer connections of Eth2 Clients

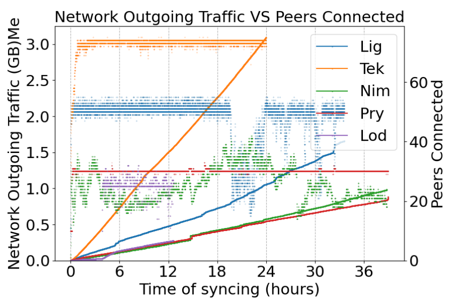

Fig. 3: Clients outgoing network traffic and connected peers

to catch up and sync faster than the other clients with a steady progress throughout the whole test. Nimbus and Lodestar seem to be the ones that take more time to sync. More importantly, it can be seen that all clients experience a sharp increase in syncing speed around slot 100,000. The fact that all clients observe the same change of steepness in the syncing curve around this slot might suggest that this phenomena is inherited from the network conditions (See Section IV-H).

B. *Peer connections*

We kept track of the number of peer connections for each client during the experiment. While this is a parameter that can be tuned, we used the default values for each client, as this is what most user would do on a regular basis. We can see in Figure 2 that Teku connects to the highest number of peers (i.e., over 70 peers) than any other client and keeps a stable number of connections during the entire experiment. Lighthouse is the client that connects with more peers after Teku with about 50 peers. After 20 hours of execution, the number of connected peers drastically decreased (oscillating around 25 peers), and later rises its peer connection back to 50 peers (hour 24). After hour 24 Lighthouse experienced sharp drops and recoveries in peer connections. Note that Lighthouse was run, from November 2nd to November 4th, while the Medalla Testnet was experiencing some issues related to non-finality [21] which might explain the peer connection drops as well as other strange phenomena observed in the Lighthouse run IV-H. Nimbus peers connections oscillate significantly, usually connecting with over 15 peers and arriving up to 42 peers. Prysm distinguishes itself for its stability of peer connections, as it succeeds in achieving 30 peer connections with almost no variation. For Lodestar, the client does not report the number of peers until the moment in which it retrieves the genesis block (hour 4), and from this point Lodestar's peer connections oscillate in the range of 20-30 peers.

C. *Outgoing network traffic*

We also monitored the outgoing network traffic for all clients. In general, we can see in Figure 3 a steady constant increase across all the clients for network outgoing data. Prysm and Nimbus are the most conservative in outgoing network traffic and their behaviour is quite similar (as their network outgoing traffic almost overlaps). Teku seems to share more data than the other clients. In general, the amount of data shared by clients seems to correlate well to the number of peer connections: clients with more peer connections have more outgoing traffic as shown in Figure 3.

D. *Incoming Network Traffic*

For incoming network traffic, we see a much less stable behaviour than for outgoing. We can see in Figure 4 that Nimbus and Prysm are again the lightest clients in terms of incoming network traffic. Prysm and Teku show a constant steady behaviour but Nimbus and Lighthouse do not.

Nimbus presents an unexpected behaviour. As shown in Figure 5, there is a point in which incoming traffic increases substantially, while at the same time the curve that represents its slot synchronization slows down. Teku is the client that has more network traffic for almost all of its execution, but Lighthouse sharply increases network incoming data from hour 20 to hour 24, overtaking Teku. To analyse this phenomena in more detail we plot the net incoming data vs the number of peer connections in Figure 6. It can be seen a considerable difference in Lighthouse's incoming network traffic, as we can witness the acceleration in receiving data from 250MB/h (hours 15-17) to 1700MB/h (hours 21-23). We notice that during those hours (i.e., 20-24) the number of peer connections decreases by about 50%, while the network incoming data increases by a factor of 6.8x. A similar (but smaller) effect is observed in hours 30-32, followed by a period of stability.

Such a behaviour could be reasonably explained by the non-finality period experienced at the end of October and beginning of November, which fits well with the important number of clients that drop out of the network during those hours. This demonstrate, how non-finality periods can impact the clients, and how these alterations can be observed with this metric monitoring. .

E. *Memory*

When it comes to memory consumption, we can see in Figure 7 how Teku and Prysm tend to have a similar behaviour, by starting with a steep usage of memory consumption and settling with a similar high memory consumption in the long

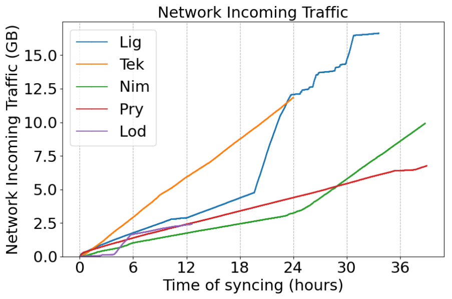

Fig. 4: Incoming network traffic of the clients

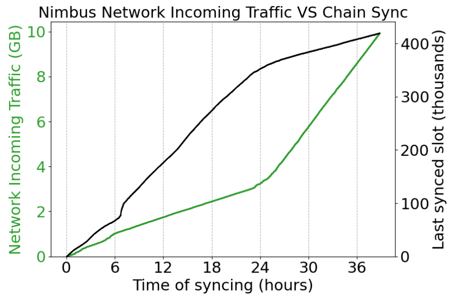

Fig. 5: Nimbus incoming traffic (green) and syncing (black)

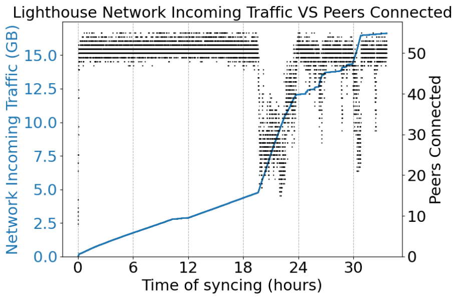

Fig. 6: Lighthouse incoming traffic (blue) and peer connections (black)

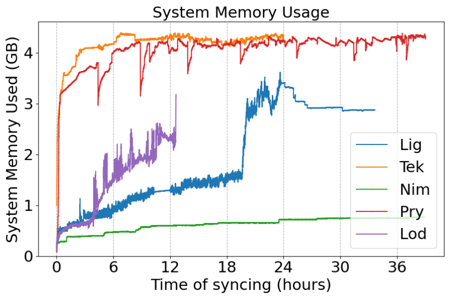

Fig. 7: Memory consumption of Eth2 clients

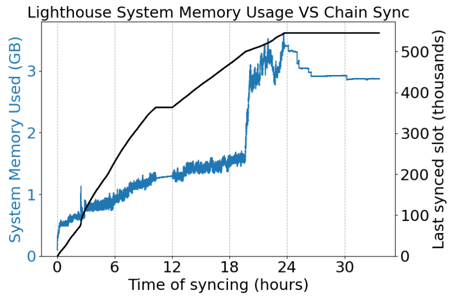

Fig. 8: Lighthouse memory (blue) and synchronization (black)

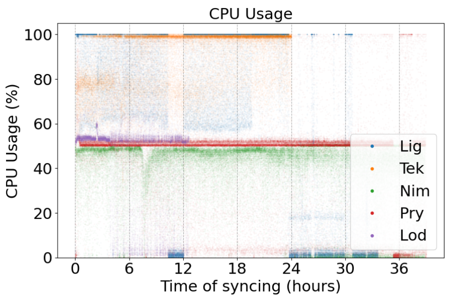

Fig. 9: CPU usage of Eth2 clients

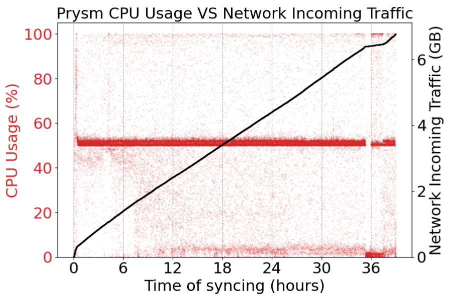

Fig. 10: Prysm CPU (red) and incoming traffic (black)

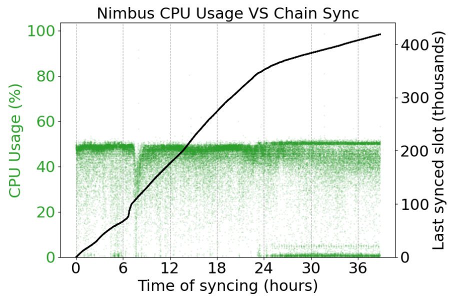

Fig. 11: Nimbus CPU (green) and slot synchronization (black)

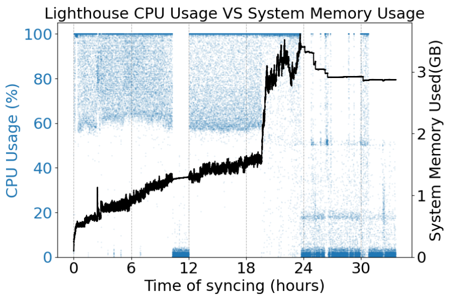

Fig. 12: Lighthouse CPU (blue) and memory usage (black)

run. Interestingly, we notice sharp drops of memory usage for Prysm at somehow regular intervals, indicating some kind of periodic cleaning process, while the behaviour of Teku is more constant. Nimbus distinguishes itself with a very low memory consumption with minimal changes. Lodestar memory usage is characterized by periodic spikes, and appears to be less demanding than the memory usage of Teku and Prysm. It can also be noticed that the first relevant spike corresponds to the moment in which the client actually starts the syncing process. Lighthouse memory consumption is low overall, but it showed some unstable behaviour.

In Figure 8 we can witness a considerable rise in memory usage for Lighthouse (hour 20) that corresponds to the time in which Lighthouse began to sharply increase its incoming network traffic and the decrease in peer connections. The memory usage oscillates around 3GB for several hours, until the 24th hour where it can be seen that Lighthouse's memory usage begins to diminish and then becomes flat. This is due to the storage limitations in the node (See Section IV-G.

F. CPU

From monitoring the CPU usage across the different clients, we noticed in Figure 9 that Nimbus has the lowest CPU consumption overall, staying always under 50%. The CPU usage of Prysm and Lodestar concentrates slightly above 50%, while the CPU usage of Lighthouse is most of the time between 60% and 100% and Teku constantly oscillates near 100%. Teku's CPU usage is always high and stable in our experiments. There might be several reasons for this. First, we performed our experiments on a virtual machine with just one core of an Intel(R) Xeon(R) E5-2620 CPU (See Section III), which might be significantly more limited than

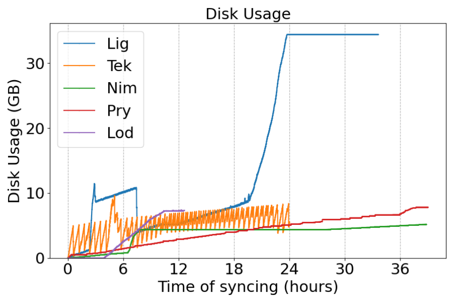

Fig. 13: Disk Usage of Eth2 clients

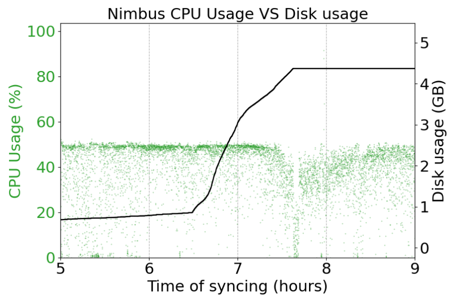

Fig. 14: Nimbus CPU (green) and slot synchronization (black)

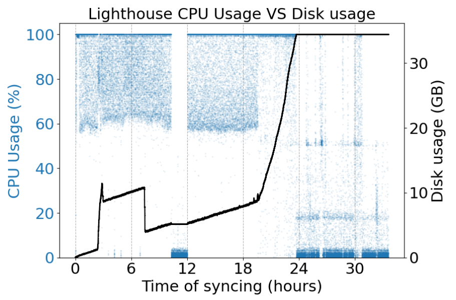

Fig. 15: Lighthouse CPU (blue) and disk usage (black)

a contemporary node with a more recent CPU generation and multiple cores. Second, we limited the JVM heap size to only 2GB because larger JVM heap sizes kept leading to crashes. With 2GB for the JVM heap, Teku managed to run smoothly without issues. However, 2GB is quite tight for Medalla testnet and it is possible that a significant portion of the time was spent in garbage collection, leading to the observed high CPU utilization.

Prysm also shows a relatively constant CPU utilization except for hour 35, where it drops to 0%, as shown in Figure 10. During that time, we also observe a strong reduction on network incoming traffic, which explains the drop on CPU utilization. It is interesting to notice that this happens even while Prysm keeps stable peer connections (See Figure 2).

Nimbus also showed a constant CPU utilization most of the time, except for a short period of time around hour 7 where it witnessed a quite sharp drop in CPU usage, as it can be seen in Figure 11. We also can observe that this drop comes just after a steep increase in blocks synced. In the other clients, such an increase in synced blocks is not followed by a CPU drop. This drop in CPU usage will be further investigated later (See Section IV-G).

Lighthouse showed some irregular CPU utilization pattern with a sudden drop to 0% CPU utilization accompanied with a flat memory usage during hours 10-12 as shown in Figure 12. The same happens after hour 24, where the client stops syncing and the CPU utilization spend most of the time near 0%. After a sharp increase in memory usage during the previous period of dropping peer connection (See Section IV-E), the memory starts to decrease gradually, ending on a flat line.

G. Disk Usage

We plot the disk usage of all clients in Figure 13. This plot shows some unexpected behaviour for some of the clients. The most stable behaviour, in terms of disk usage among all clients, seems to be Prysm according to this experiment. Nimbus maintains a low storage usage for most of the time, although it does experience a sharp increase in storage at some point. We investigate this increase in storage by isolating Nimbus' disk utilization and comparing it with its CPU usage. We can see in Figure 14 that the increase in storage is quite dramatic, it goes from 1GB to over 4GB in a short period of time. Curiously, just after this storage increase, the CPU usage of Nimbus drops substantially. While it is true that periods of high storage intensity are often followed by idle CPU time. Such idle CPU times just last for milliseconds or seconds after storage bursts, so this solely it is unlikely to explain the whole drop of CPU utilization. In addition, this is not observed in other clients that also experience large increases in disk utilization.

Lighthouse also shows a steep increase in disk usage, although much more important than Nimbus, going from 3GB to over 10GB in minutes. After this, the disk usage continues to increment at the same speed than initially doing, keeping the same growth rate for several hours, to then drop sharply at around hour 8. Interestingly, extrapolating the initial storage growth rate would place Lighthouse at the same spot after 9 hours of syncing, as if the whole storage used during the perturbation was just temporary data that once everything solved, could be discarded without losing any real information. After 20 hours, however, Lighthouse observes another sharp increase in disk usage, but this time it keeps going up until it consumes the whole 34GB of space in the node, at which point it becomes a flat line for the next 10 hours.

In figure 15, we compare the disk and CPU utilization of Lighthouse. As we can see, in the same spots where there is no CPU utilization (i.e., hours 10-12 and 24-34) the disk utilization also becomes flat, showing that the client stall during those last hours.

This phenomena, could be linked to a Medalla non-finality period during October-November caused by a unusually low number of validators [21]. With the exception of Lighthouse's behaviour, caused by a consensus problem in the Medalla Testnet, Lodestar seems to use more storage than all the other clients. In addition, we noticed that Lodestar starts gathering data after a long period of time (4 hours), which seems to be linked to the time spent getting the genesis block.

Teku, on the other hand, demonstrated a strange disk usage pattern that resembles to a zig-zag as shown in Figure 13. This zig-zag pattern could be explained by the fact that Teku uses RocksDB, which sometimes implements a periodic compression scheme in which the database grows for a while and then compacts itself. Nonetheless, its periodicity gets disrupted at some point, and from hours 4 to 6 we can see

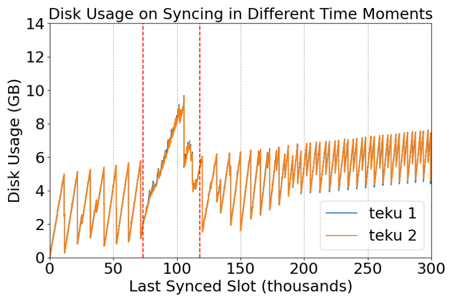

Fig. 16: Teku Disk Usage for two different executions

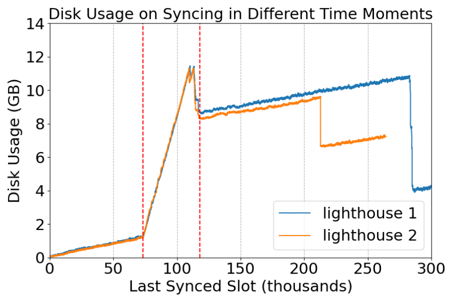

Fig. 17: Lighthouse Disk Usage for two different executions

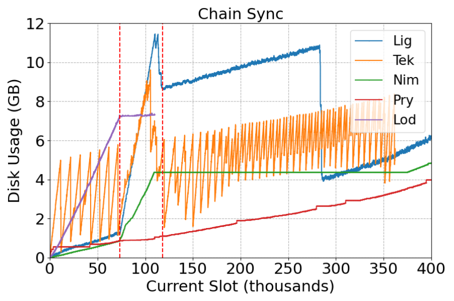

Fig. 18: Disk Usage of Eth2 clients for slots

an important perturbation in the disk utilization.

To have a better view of the perturbation, we plotted the disk utilization of Teku by last synced slot (instead of wall-clock time) and we can clearly see that the perturbation happens between slots 70,000 and 120,000, as shown in Figure 16. Given that this perturbation could be related to some random internal process in the node or events linked to RocksDB or the JVM, we decided to run Teku again from scratch on the same node. To our surprise the disk usage pattern of the second run is almost identical to the first one, with the same perturbation between the same slots (i.e., 70,000-120,000). Note that these two runs were done with a distance of 3 days between them. This deterministic behaviour implies that the client is simply requesting and processing blocks in order, one after the other, therefore the client is syncing through a finalised part of the chain where there are no forks to explore and everything is deterministic. We also ran Lighthouse twice to check if the disk behaviour was deterministic and it is for most of the period between slots 75,000 and 120,000, as shown in Figure 17. However, Lighthouse behaviour does change a bit after slot 210,000: the second run managed to reduce storage usage way before the first run. Although, this shows that this behaviour is not just an artifact of our experiments, this does not explain why the disk grows so fast and what is the root-cause of the perturbation.

H. *Fossil Records of Non-Finality*

In an attempt to compare the disk behaviour of all clients in a synchronous way, we have plotted the storage usage of all clients with respect to the slot number in Figure 18. In this plot, we see that almost all clients change behaviour somewhere around slot 70,000 and go back to a normal behaviour around slot 120,000. This period corresponds with the non-finality period of Medalla. Indeed, the Medalla network passed through a period of non-finality caused by erroneous rough time responses witnessed by Prysm clients [22].

During this period, we observe a sharp increase in the disk usage of Teku, Lighthouse and Nimbus, while Lodestar shows the exact opposite and the behaviour of Prysm seems to be unaffected. For Lighthouse and Teku, we can see that the sharp increase in disk usage is followed by a similarly sharp drop towards the end of this period. Considering the relatively similar behaviour of both clients, it is interesting to notice how Teku reduced the time of higher disk usage, while Lighthouse keeps running 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 BeaconState objects, and a *cold* database that stores finalized BeaconStates. 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.

On the other hand, Nimbus rise in disk storage is not as sharp as Teku and Lighthouse, however it did not reduce its storage afterwards (in contrast to Teku and Lighthouse). Oddly, we can notice that Lodestar's disk usage increases more rapidly than any other client at the beginning, until the start of this non-finality period, when it stops growing at all. 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 in intervals of 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 growing unnecessarily large. However doing this comes at a cost, as they now keep everything in memory so if they do need to retrieve a particular state (unfinalized) 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 and can make keeping track of all the different forks harder.

During this non-finality period, clients also showed a steeper syncing curve around slot 100,000, as we can see in Figure 19. This could imply that during this period there is little information to process and 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 Teku logs to analyse the number of times a block was queued and/or processed for each slot during the non-finality period. The results, depicted in Figure 20, show that during this period there were almost no blocks queued, which seems to be consistent with the accelerated syncing speed. However we also notice that just at the beginning of the non-finality period, at exactly slot 73,248, there were 219 blocks queued (note the logarithmic Y axis). This clearly shows a huge perturbation in the network.

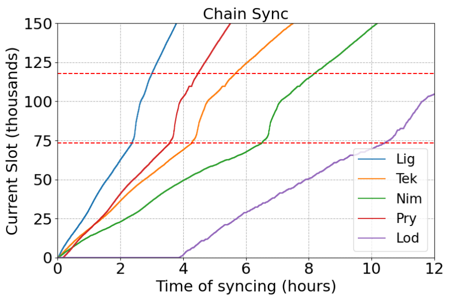

Fig. 19: Detail of slot synchronization of Eth2 clients

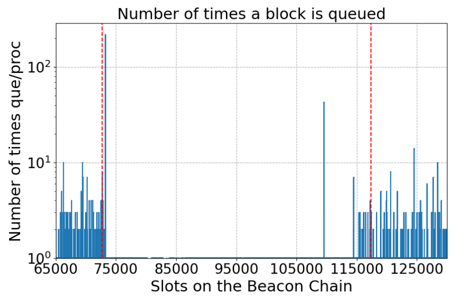

Fig. 20: Number of times a block is queued

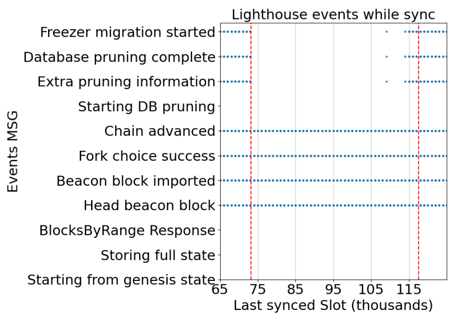

Fig. 21: Events timeline for Lighthouse

We assume that the accelerating disk usage is related to an increase in the state stored in the database of the client, and this might be linked to a difficulty of pruning states during a non-finality period. Thus, to corroborate our hypothesis, we analysed Lighthouse detailed logs and plot the frequency at which different events get executed. Figure 21 lists a total of 11 different types of events. We can see that during the non-finality period there are four type of events that almost never get executed: *Freezer migration started, Database pruning complete, Extra pruning information, and Starting database pruning*. This demonstrates that during this period the client is unable to prune the database, which is consistent with the rapid disk usage increase.

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

V. Discussion

As we could see in section IV, despite sharing the same Eth2 specification, the five clients showed completely different implementations and, therefore, different behavior. These substantial differences between them emerge from the whole decentralization objective of Eth2. Eth2 proposes a new method to achieve scalability while keeping the security and decentralization aspects of the blockchain. With this motivation in mind, Eth2 tries to offer several possibilities of client implementations to participate in the network. Increasing the accessibility and, therefore, the number of users allows the technology to upscale without compromising security or decentralization.

From optimized embedded system-oriented client as Nimbus to high-performance business orientation of Teku, the selected five Eth2 clients represent the entire spectrum of users and hardware that can participate in the network.

Among the clients, we have seen how Nimbus performs in a low resources situation. This kind of client implementation is optimized for embedded systems. Nimbus achieves a surprising performance on the synchronization time, and it achieves this while requiring only limited hardware resources (CPU and memory). This client implementation has been tested on a Raspberry Pi 3, and it really aims to be the choice of those who want to participate in the network without dedicating too many resources into it.

On the other side of the spectrum, we have the business performance-oriented client Teku. Teku did not show enough solvency when working with limited resources. On the contrary, it was relatively heavy for our testing hardware. However, Teku does show high availability as it connects with many peers and receives and processes much more data than the other clients.

In the case of Lighthouse, the team has a strong focus on the reliability of the client. For instance, Lighthouse is the only client that tracks all the available forks in a moment of network instability. This choice serves the entire information need to perform the validator duties, minimizing economic slashing. Lighthouse also implements a double database strategy to work with finalized and non-finalized states. However, it is essential to keep in mind the disk requirements for those strategies. Otherwise, the client would be exposed to possible failures due to storage limitations.

Prysm seems to be the most mature project among the Eth2 clients. It has shown stability and robustness while achieving a fast synchronization. Its user accessibility, the multipurpose hardware compatibility, as the stable performance makes it the most complete client among the tested one. Ironically, a timing bug coming from the Prysm client implementation was the root cause of the Medalla testnet, which was observed by our study.

Lodestar is probably the least mature of the clients, which was observed on frequent crashes at the time of testing. It can be run with relatively moderate resources. However, the steep outgoing network traffic and the time waited in order to get the genesis block leave space for optimization.

VI. Conclusion

In this work, we have performed a detailed analysis of the resource usage of Eth2 clients. Through this analysis, we have noticed the effort of different teams to develop functional software to tackle the transitioning to Eth2. The necessity of relying on several types of clients is reflected by the different objectives and resources of a wide variety of users. Our analysis showed significant differences between clients in terms of CPU, memory, and disk utilization. We also observed some network perturbations where the number of peers dropped quickly and its impact on the other resources. To the best of our knowledge, this is the first study of the Eth2 clients' resource usage at this scale and demonstrates that it is possible to detect non-finality periods by simply looking at the clients' resource usage. The diverse behavior of the analyzed clients is a positive sign for the Ethereum community, as it shows a solid variety of software available for Eth2 validators, and this provides the network with further decentralization, resilience, and security. However, significant efforts need to be made to correct some of the issues observed in some of the clients during this study.

In future work, we would like to study further how non-finality periods affect the resource usage of Eth2 clients to implement online detectors that use this information to signal possible network perturbations in real-time.

Acknowledgment

This work has been supported by the Ethereum Foundation under Grant FY20-0198. We would like to thank the researchers of the Ethereum Foundation for their feedback and suggestions. We would also like to thank the developers and researchers of the five client teams for all their help to set up our experiments and their constructive feedback on this study.

References

[1] "Ethereum whitepaper," https://ethereum.org/en/whitepaper/.

[2] "Phase-0," https://notes.ethereum.org/@djrtwo/Bkn3zpwxB.

[3] M. Bez, G. Fornari, and T. Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, April 2019, pp. 167–176.

[4] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 91–104. [Online]. Available: https://doi.org/10.1145/3278532.3278542

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

[6] O. Moindrot and C. Bournhonesque, "Proof of stake made simple with casper," *ICME, Stanford University*, 2017.

[7] D. Sel, K. Zhang, and H.-A. Jacobsen, "Towards solving the data availability problem for sharded ethereum," in *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, ser. SERIAL'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 25–30. [Online]. Available: https://doi.org/10.1145/3284764.3284769

[8] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, no. 3, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3391195

[9] D. Park, Y. Zhang, and G. Rosu, "End-to-end formal verification of ethereum 2.0 deposit smart contract," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 151–164.

[10] S. Rouhani and R. Deters, "Performance analysis of ethereum transactions in private blockchain," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, pp. 70–74.

[11] "Medalla test network:," https://github.com/goerli/medalla.

[12] "Monitoring scripts:," https://github.com/leobago/BSC-ETH2/tree/master/ETH2-clients-comparison/scripts/parsing_scripts/metrics_monitoring_scripts.

[13] "Logs parsing scripts:," https://github.com/leobago/BSC-ETH2/tree/master/ETH2-clients-comparison/scripts/parsing_scripts/client_logs_analysis_scripts.

[14] "Plotting scripts:," https://github.com/leobago/BSC-ETH2/tree/master/ETH2-clients-comparison/scripts/plotting_scripts.

[15] "Teku version:," https://github.com/ConsenSys/teku/commit/6883451c9f87a28c5923dbc8f291db237930cad0.

[16] "Prysm version:," https://github.com/prysmaticlabs/prysm/commit/4bc7cb6959a1ea5b4b4b53b42284900e3b117dea.

[17] "Lighthouse version:," https://github.com/sigp/lighthouse/commit/95c96ac567474df2abb4e9da9f5e771cf5a7426d.

[18] "Nimbus version:," https://github.com/status-im/nimbus-eth2/commit/9255945fb0ef1bf036d32b7cce5df42a8dd69be7.

[19] "Lodestar version:," https://github.com/ChainSafe/lodestar/pull/1731.

[20] "Multi-client benchmark on medalla testnet 2020/10/01:," https://github.com/q9f/eth2-bench-2020-10.

[21] "Medalla non-finality october 2020:," https://gist.github.com/yorickdowne/ea9b18ac2b51a508080c9a810d978522.

[22] "Medalla non-finality period august 2020:," https://docs.google.com/document/d/11RmitNRui10LcLCyoXY6B1INCZZKq30gEU6BEg3EWfk.